# MINI PROJECT REPORT

## Objectives Of Project

- Develop programs for complex real world problems.
- Apply good programming practices in their code like Comments, indentation etc.
- Utilize Debugger and its tools like gdb/gnu for error handling.
- Demonstrate configuration and usage of different software tools used in industry.

# Function Description

## Function 1 - hcf

First function is hcf which takes 2 arguements of datatype int and prints the hcf of both the entered numbers.
Hcf is calculated by subtracting the smaller number from the larger one and continuing this process until both the numbers became equal.
The value at which both the numbers became equal is the required hcf.

## Function 2 - palindrome

Second function is palindrome which take 1 arguement of datatype int and prints if the entered number is a palindrome or not.
A number is said to be palindrome if its original value is equal to its reverse value.

## Function 3 - perfect

Third function is perfect which take 1 arguement of datatype int and prints if the entered number is a perfect number or not.
A number is said to be perfect if sum of its divisors ,except the number itself, is equal to the number.

## Function 4 - prime

Fourth function is prime which take 1 arguement of datatype int and prints if the entered number is a prime number or not.
A number is said to be prime if it is divisible by only 1 and the number itself.

## Function 5 - armstrong

Fifth function is armstrong which take 1 arguement of datatype int which is a three digit integer and prints if the entered number is a armstrong number or not.
A three digit number is said to be armstrong number if sum of cubes of its digit is equal to the original number.

## Function 6 - factorial

Sixth function is factorial which take 1 arguement of datatype int and prints factorial of entered number.
Factorial of any number n is the product of first n natural numbers.

## Function 7 - sum

Seventh function is sum which take 1 arguement of datatype int and prints the sum of first n natural numbers.

## Function 8 - coprime

Eighth function is coprime which takes 2 arguement of datatype int and prints if enetered numbers are co-prime to each other or not.
Two numbers are said to be co-prime if they have only one factor in common, that is, 1.

## Function 9 - power

Ninth function is power which take 2 arguement of datatype int ,that are base and exponent, and prints the value of base to the power exponent.

## Function 10 - magic_number

Tenth function is magic_number which take 1 arguement of datatype int and prints if the entered number is magic number or not.
A number is said to be magic number if we sum its digits and again sum the digits of the number obtained until we reach a single digit.
If the single digit obtained is 1 the number is said to be as magic number, otherwise the number is not a magic number.

# Codes

Code screenshots and Output in C++

parmeet@LAPTOP-QHS752JI: ~

GNU nano 4.8 minip

```
#include<iostream>
using namespace std;
/*This programme will coantain the operations which
we can perform of numbers*/
void hcf(int number_1, int number_2)      /*declaration of function*/
/* This function is used to calculate the HCF of 2 numbers*/
{ while (number_1 != number_2){           /*Loop used to calculate hcf*/
    if (number_1>number_2){
       number_1 = number_1 - number_2;
    }else{
        number_2 = number_2 - number_1;
    }
}
  cout<<"HCF of the numbers is "<<number_1<<endl;   /*printing hcf*/
}

 void palindrome(int number_1)
 /*This fumction is used to check wether the
 number is palindrome or not*/
 { int reverse = 0;          /*this is uesd to store reverse of that number*/
   int reminder;             /*this is used to store reminder*/
   while (number_1 > 0){     /* loop for finding out wether the number is palindrome or not*/
    reminder = number_1 % 10;
    reverse = (10 * reverse) + reminder;
    number_1 = number_1 / 10;
   }
    if (number_1==reverse){
        cout<<"This number is palindrome"<<endl;
    }else{
        cout<<"This number is not a palindrome"<<endl;
    }
 }

 void perfect(int number_1)
 {
    /*This function is created to find out whether the number
    is perferct number or not*/
    int sum = 0;
    for (int counter=1; counter<number_1; counter++){   /*Loop to calculate sum f divison*/
        if (number_1 % counter == 0){
            sum = sum + counter;
        }
    }
    if (sum == number_1){                           /* printting perfect number */
        cout<<"The number is  perfect"<<endl;
    }else{
        cout<<"The number is not perfect"<<endl;
 }
```

parmeet@LAPTOP-QHS752JI: ~

GNU nano 4.8

```
        cout<<"The number is not perfect"<<endl;
    }
}

void prime(int number_1){
    /* function to calculate whther the number is prime or not*/
    int flag = 0;
    for (int counter = 2; counter<number_1; counter++){    /*logic of prime number*
        if (number_1 % counter == 0){
            flag++;
            break;
        }
        if (flag == 0){                                    /*priting prime number*/
            cout<<"The number is  prime"<<endl;
        }else{
            cout<<"The number is not prime"<<endl;
        }
    }
}

void armstrong(int number_1)
{  /* function for checking whether the number is armstrong or not*/
    int sum = 0;
    int digit;
    int duplicate;
    duplicate = number_1;
    while (number_1 > 0){        /*logic for ar,strong number*/
        digit = number_1 % 10;
        sum = sum + (digit * digit * digit);
        number_1 = number_1 / 10;
    }
    if (sum == duplicate){        /* printting armstrong number*/
        cout<<"Number is armstrong"<<endl;
    }else{
        cout<<"Number is not armstrong"<<endl;
    }
}
void factorial(int number_1)
{  /* function for printinf factorial of the given number*/
    int fact = 1;           /* logic of calcultaing factorial*/
    for (int counter = 2; counter <= number_1; counter++){
        fact = fact * counter;
    }
    cout<<fact<<endl;
}

void sum(int number_1)
{   /*function  for printing sum uptil number n*/
```

parmeet@LAPTOP-QHS752JI: ~

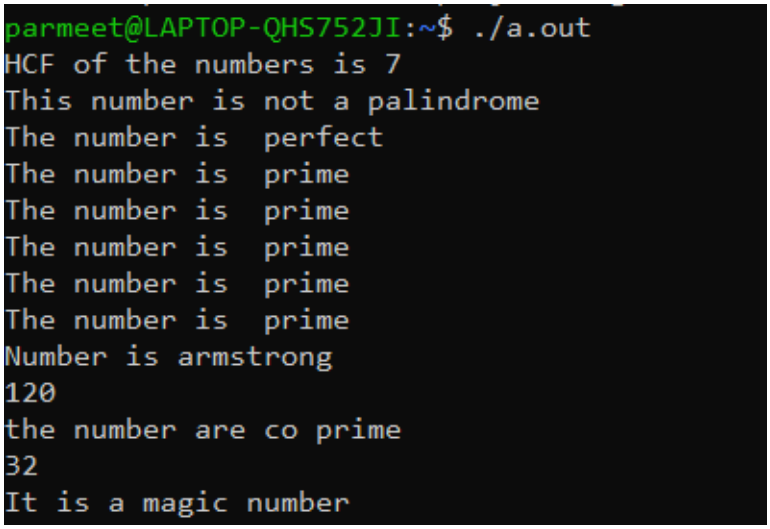
GNU nano 4.8

```
}

void sum(int number_1)
{   /*function  for printing sum uptil number n*/
   int sum = 0;
   for (int counter=1; counter <= number_1; counter++){  /*logic for calculating s
   sum = sum + number_1;
   }
   cout<<"Sum upto number_1 is "<<sum<<endl;
}

void coprime(int number_1,int number_2)
{  /*to check whether the given numbers are co prime or no*/
   for (int counter = 2; counter < number_1 && counter < number_2; counter++){ /*
       if (number_1 % counter == 0 && number_2 % counter == 0){
           cout<<"The number are not coprime "<<endl;
           return;
       }
   }
   cout<<"the number are co prime "<<endl;
}

void power(int base, int exponent)
/* Function to print power of a given number*/
{  int answer = 1;            /*flag for calculating power*/
   for (int counter = 1; counter <= exponent ; counter++){       /*Loop for calcul
       answer = answer * base;
   }
   cout<<answer<<endl;                          /*printting answer*/
}
void magic_number(int number_1)
{  /* function for checking whather the number is magic nmber or not*/
   int digit,sum=number_1;
   while (number_1 >= 10){         /*logic for calculating magic number*/
       sum = 0;
       while (number_1 != 0){
           digit = number_1 % 10;
           sum = sum + digit;
           number_1 = number_1 / 10;
       }
       number_1 = sum;
   }
   if (sum == 1){
       cout<<"It is a magic number "<<endl;
   }else{
       cout<<"It is nit a magic number"<<endl;
   }
```

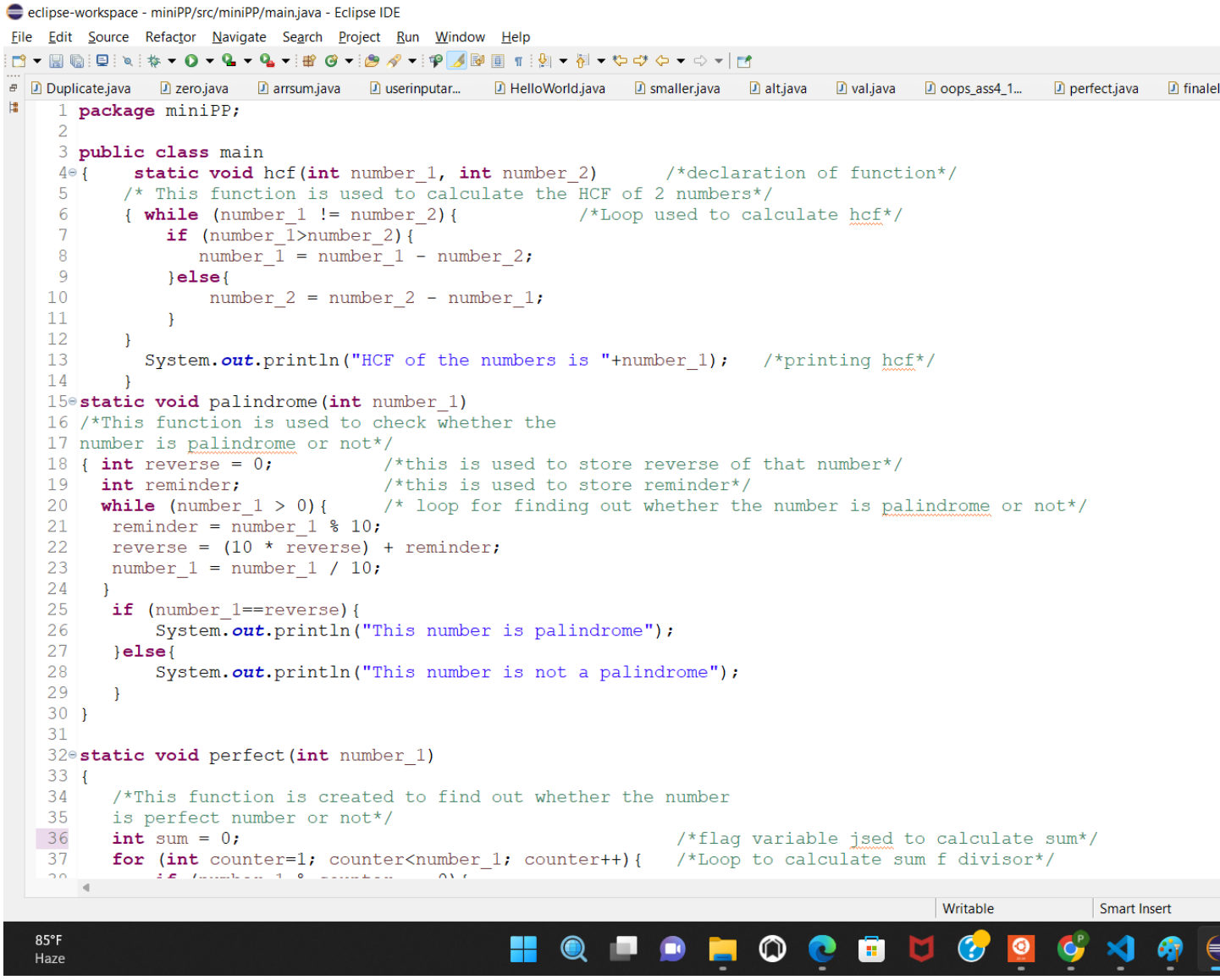
parmeet@LAPTOP-QHS752JI: ~

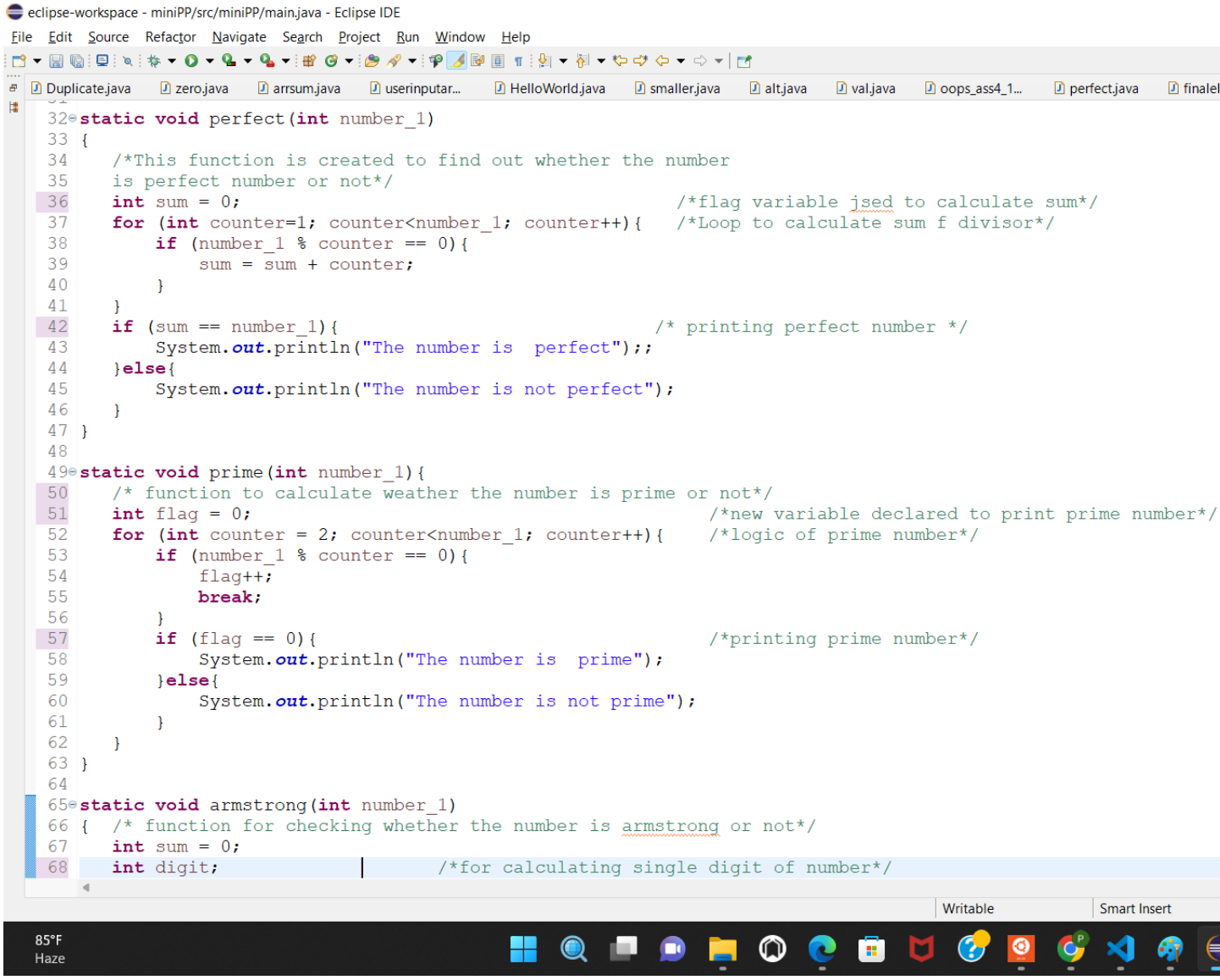
GNU nano 4.8

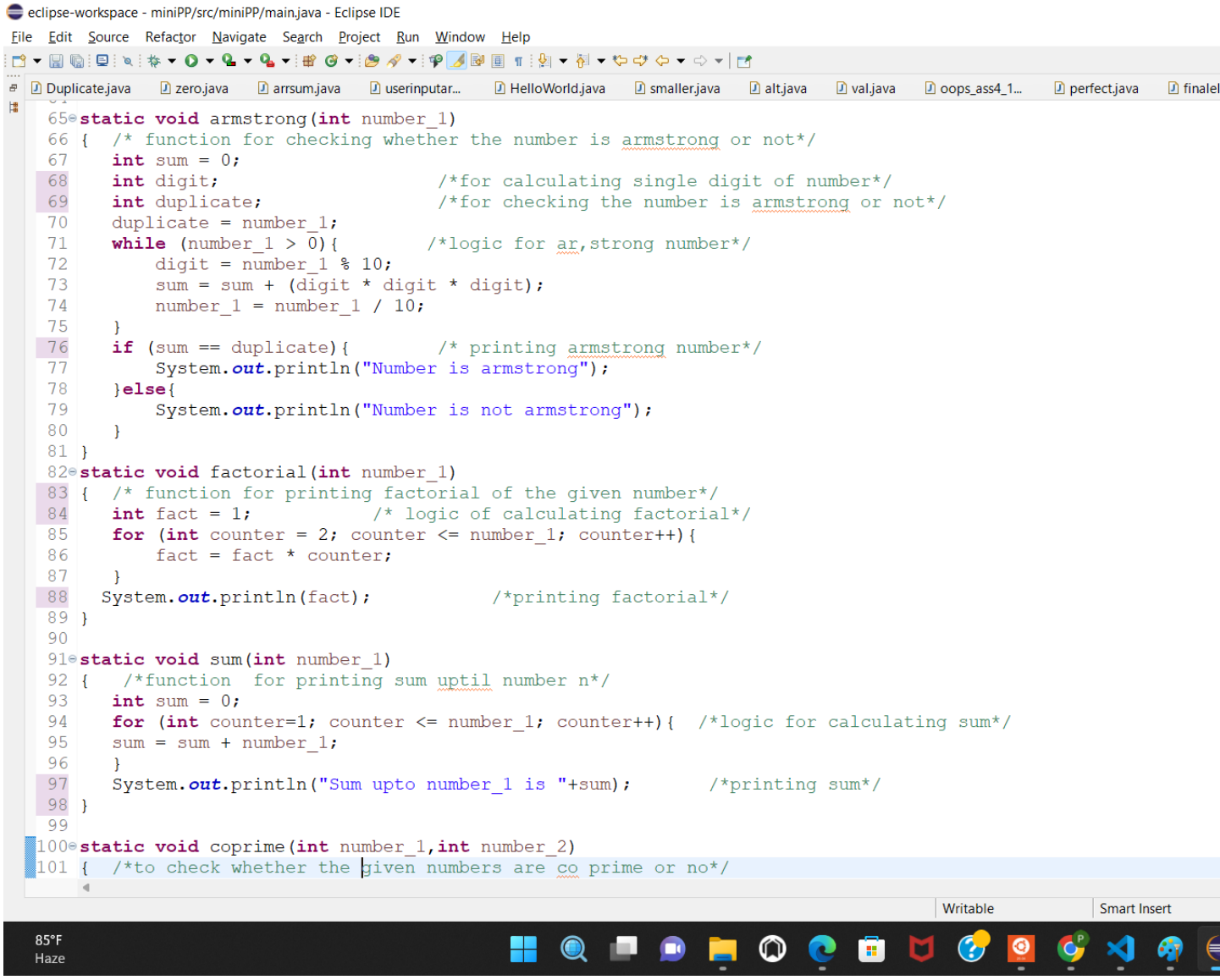
```
    }
    cout<<"the number are co prime "<<endl;
}

void power(int base, int exponent)
/* Function to print power of a given number*/
{   int answer = 1;             /*flag for calculating power*/
    for (int counter = 1; counter <= exponent ; counter++){       /*Loop for calcul
        answer = answer * base;
    }
    cout<<answer<<endl;                             /*printting answer*/
}
void magic_number(int number_1)
{   /* function for checking whather the number is magic nmber or not*/
    int digit,sum=number_1;
    while (number_1 >= 10){         /*logic for calculating magic number*/
        sum = 0;
        while (number_1 != 0){
            digit = number_1 % 10;
            sum = sum + digit;
            number_1 = number_1 / 10;
        }
        number_1 = sum;
    }
    if (sum == 1){
        cout<<"It is a magic number "<<endl;
    }else{
        cout<<"It is nit a magic number"<<endl;
    }

}


int main()
{
    /* implementation of all functions*/
hcf(14,21);
palindrome(654);
perfect(6);
prime(7);
armstrong(371);
factorial(5);
coprime(21,26);
power(2,5);
magic_number(1234);
    return 0;
}
```

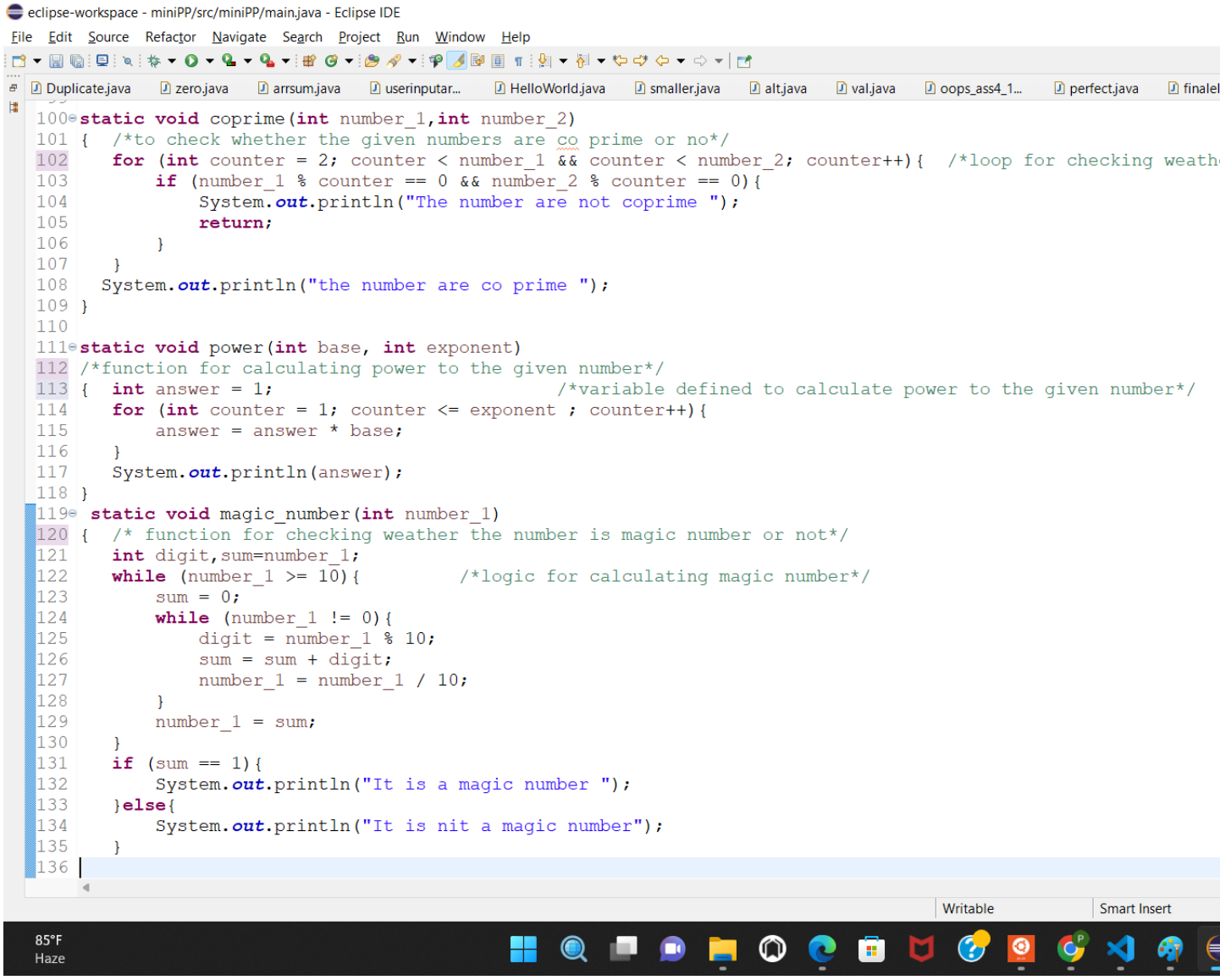


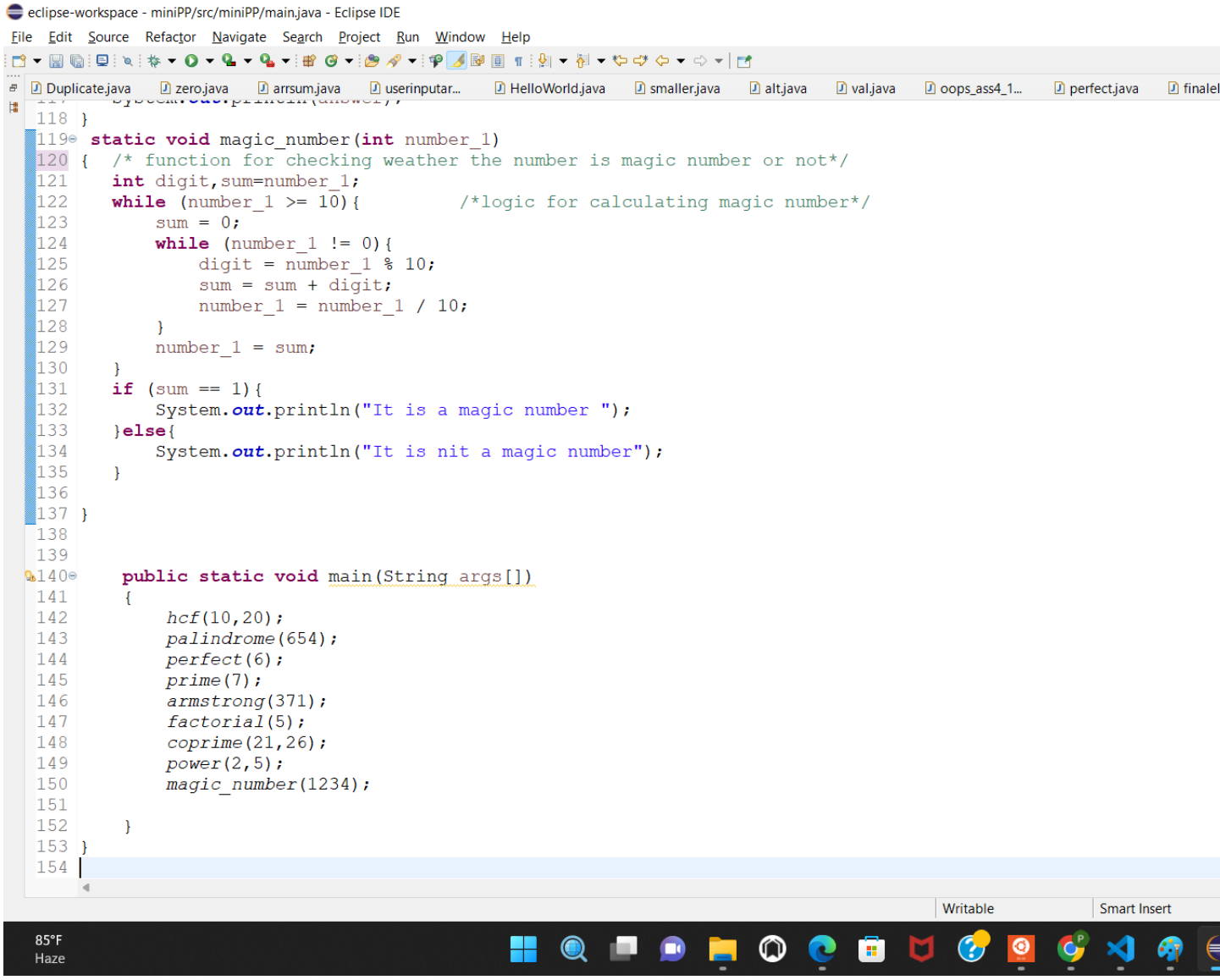

Code screenshots and Output in JAVA

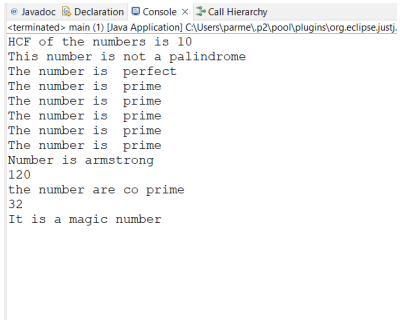

eclipse-workspace - miniPP/src/miniPP/main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Duplicate.java zero.java arrsum.java userinputar... HelloWorld.java smaller.java alt.java val.java oops_ass4_1... perfect.java finalel
1 package miniPP;
2
3 public class main
4 { static void hcf(int number_1, int number_2) /*declaration of function*/
5 /* This function is used to calculate the HCF of 2 numbers*/
6 { while (number_1 != number_2){ /*Loop used to calculate hcf*/
7 if (number_1>number_2){
8 number_1 = number_1 - number_2;
9 }else{
10 number_2 = number_2 - number_1;
11 }
12 }
13 System.out.println("HCF of the numbers is "+number_1); /*printing hcf*/
14 }
15 static void palindrome(int number_1)
16 /*This function is used to check whether the
17 number is palindrome or not*/
18 { int reverse = 0; /*this is used to store reverse of that number*/
19 int reminder; /*this is used to store reminder*/
20 while (number_1 > 0){ /* loop for finding out whether the number is palindrome or not*/
21 reminder = number_1 % 10;
22 reverse = (10 * reverse) + reminder;
23 number_1 = number_1 / 10;
24 }
25 if (number_1==reverse){
26 System.out.println("This number is palindrome");
27 }else{
28 System.out.println("This number is not a palindrome");
29 }
30 }
31
32 static void perfect(int number_1)
33 {
34 /*This function is created to find out whether the number
35 is perfect number or not*/
36 int sum = 0; /*flag variable jsed to calculate sum*/
37 for (int counter=1; counter<number_1; counter++){ /*Loop to calculate sum f divisor*/
Writable
Smart Insert
85°F
Haze


eclipse-workspace - miniPP/src/miniPP/main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Duplicate.java zero.java arrsum.java userinputar... HelloWorld.java smaller.java alt.java val.java oops_ass4_1... perfect.java finalel
static void perfect(int number_1)
{
/*This function is created to find out whether the number
is perfect number or not*/
int sum = 0; /*flag variable jsed to calculate sum*/
for (int counter=1; counter<number_1; counter++){ /*Loop to calculate sum f divisor*/
if (number_1 % counter == 0){
sum = sum + counter;
}
}
if (sum == number_1){ /* printing perfect number */
System.out.println("The number is perfect");;
}else{
System.out.println("The number is not perfect");
}
}
static void prime(int number_1){
/* function to calculate weather the number is prime or not*/
int flag = 0; /*new variable declared to print prime number*/
for (int counter = 2; counter<number_1; counter++){ /*logic of prime number*/
if (number_1 % counter == 0){
flag++;
break;
}
if (flag == 0){ /*printing prime number*/
System.out.println("The number is prime");
}else{
System.out.println("The number is not prime");
}
}
}
static void armstrong(int number_1)
{ /* function for checking whether the number is armstrong or not*/
int sum = 0;
int digit; /*for calculating single digit of number*/
Writable
Smart Insert
85°F
Haze

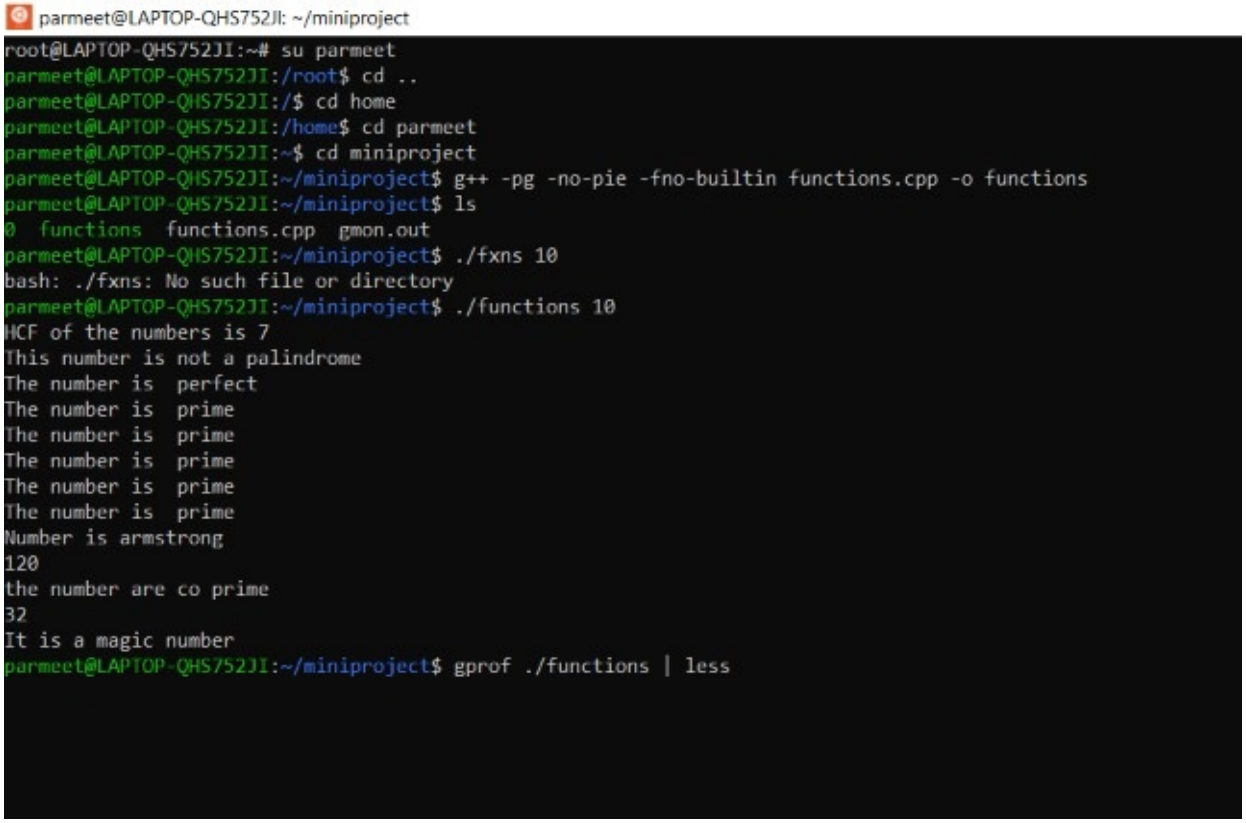

eclipse-workspace - miniPP/src/miniPP/main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Duplicate.java zero.java arrsum.java userinputar... HelloWorld.java smaller.java alt.java val.java oops_ass4_1... perfect.java finalel
65 static void armstrong(int number_1)
66 { /* function for checking whether the number is armstrong or not*/
67 int sum = 0;
68 int digit; /*for calculating single digit of number*/
69 int duplicate; /*for checking the number is armstrong or not*/
70 duplicate = number_1;
71 while (number_1 > 0){ /*logic for ar,strong number*/
72 digit = number_1 % 10;
73 sum = sum + (digit * digit * digit);
74 number_1 = number_1 / 10;
75 }
76 if (sum == duplicate){ /* printing armstrong number*/
77 System.out.println("Number is armstrong");
78 }else{
79 System.out.println("Number is not armstrong");
80 }
81 }
82 static void factorial(int number_1)
83 { /* function for printing factorial of the given number*/
84 int fact = 1; /* logic of calculating factorial*/
85 for (int counter = 2; counter <= number_1; counter++){
86 fact = fact * counter;
87 }
88 System.out.println(fact); /*printing factorial*/
89 }
90
91 static void sum(int number_1)
92 { /*function for printing sum uptil number n*/
93 int sum = 0;
94 for (int counter=1; counter <= number_1; counter++){ /*logic for calculating sum*/
95 sum = sum + number_1;
96 }
97 System.out.println("Sum upto number_1 is "+sum); /*printing sum*/
98 }
99
100 static void coprime(int number_1,int number_2)
101 { /*to check whether the given numbers are co prime or no*/
Writable
Smart Insert
85°F
Haze

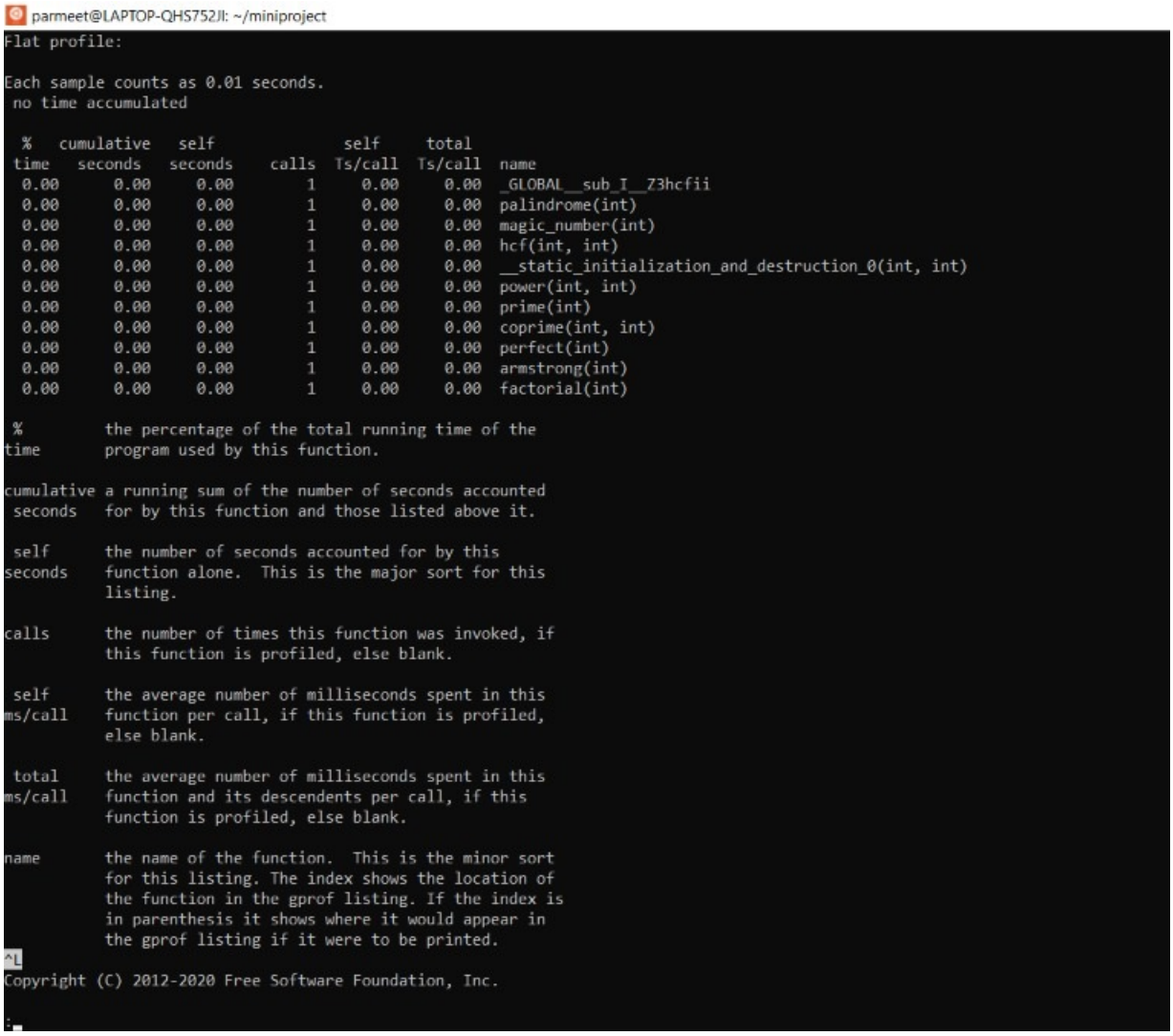

eclipse-workspace - miniPP/src/miniPP/main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Duplicate.java zero.java arrsum.java userinputar... HelloWorld.java smaller.java alt.java val.java oops_ass4_1... perfect.java finalel
100 static void coprime(int number_1,int number_2)
101 { /*to check whether the given numbers are co prime or no*/
102 for (int counter = 2; counter < number_1 && counter < number_2; counter++){ /*loop for checking weath
103 if (number_1 % counter == 0 && number_2 % counter == 0){
104 System.out.println("The number are not coprime ");
105 return;
106 }
107 }
108 System.out.println("the number are co prime ");
109 }
110
111 static void power(int base, int exponent)
112 /*function for calculating power to the given number*/
113 { int answer = 1; /*variable defined to calculate power to the given number*/
114 for (int counter = 1; counter <= exponent ; counter++){
115 answer = answer * base;
116 }
117 System.out.println(answer);
118 }
119 static void magic_number(int number_1)
120 { /* function for checking weather the number is magic number or not*/
121 int digit,sum=number_1;
122 while (number_1 >= 10){ /*logic for calculating magic number*/
123 sum = 0;
124 while (number_1 != 0){
125 digit = number_1 % 10;
126 sum = sum + digit;
127 number_1 = number_1 / 10;
128 }
129 number_1 = sum;
130 }
131 if (sum == 1){
132 System.out.println("It is a magic number ");
133 }else{
134 System.out.println("It is nit a magic number");
135 }
136
Writable
Smart Insert
85°F
Haze

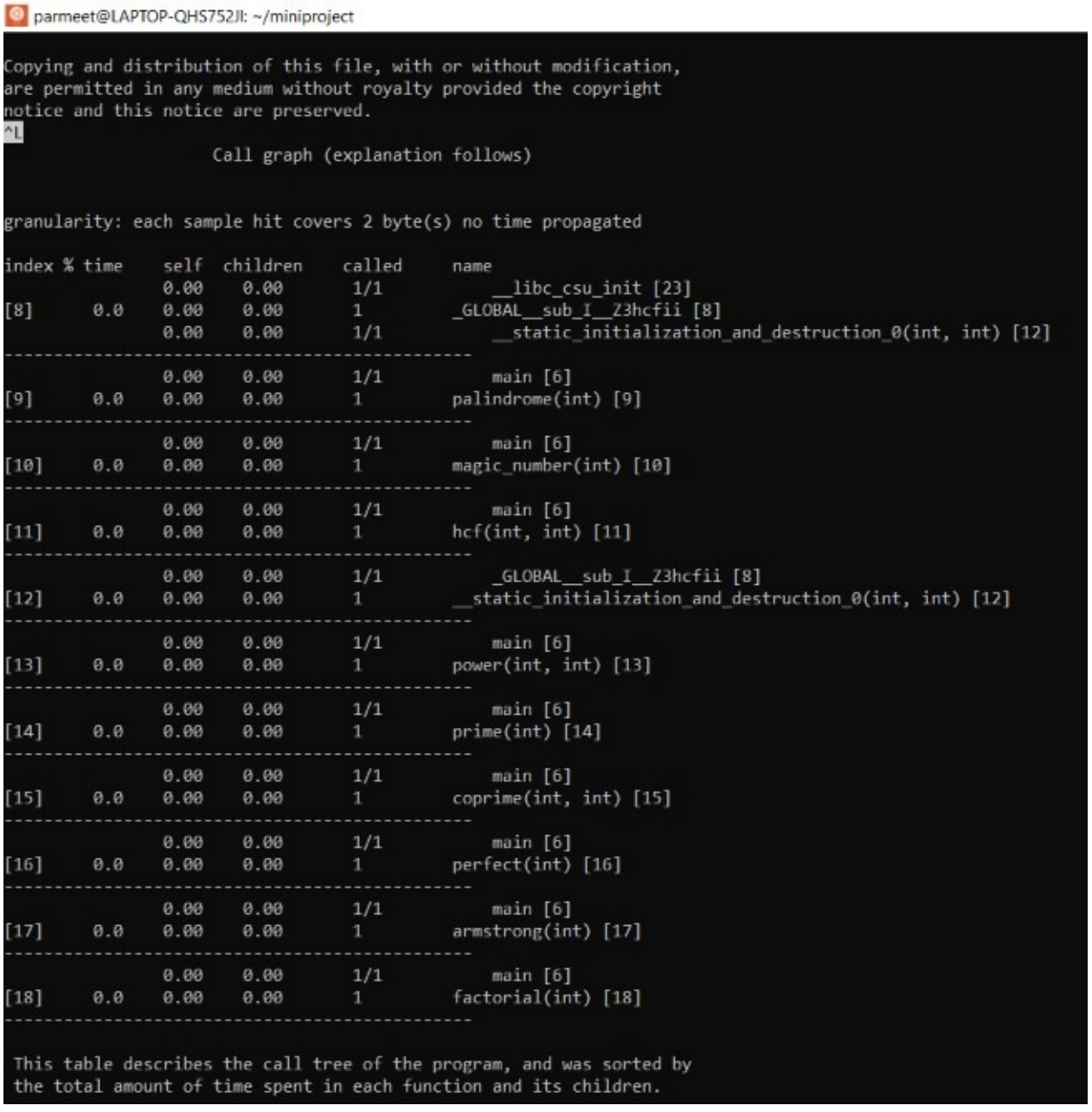

eclipse-workspace - miniPP/src/miniPP/main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Duplicate.java zero.java arrsum.java userinputar... HelloWorld.java smaller.java alt.java val.java oops_ass4_1... perfect.java finalel
118 }
119 static void magic_number(int number_1)
120 { /* function for checking weather the number is magic number or not*/
121 int digit,sum=number_1;
122 while (number_1 >= 10){ /*logic for calculating magic number*/
123 sum = 0;
124 while (number_1 != 0){
125 digit = number_1 % 10;
126 sum = sum + digit;
127 number_1 = number_1 / 10;
128 }
129 number_1 = sum;
130 }
131 if (sum == 1){
132 System.out.println("It is a magic number ");
133 }else{
134 System.out.println("It is nit a magic number");
135 }
136
137 }
138
139
140 public static void main(String args[])
141 {
142 hcf(10,20);
143 palindrome(654);
144 perfect(6);
145 prime(7);
146 armstrong(371);
147 factorial(5);
148 coprime(21,26);
149 power(2,5);
150 magic_number(1234);
151
152 }
153 }
154
Writable
Smart Insert
85°F
Haze


Javadoc Declaration Console Call Hierarchy
<terminated> main (1) [Java Application] C:\Users\parme\.p2\pool\plugins\org.eclipse.justj.
HCF of the numbers is 10
This number is not a palindrome
The number is perfect
The number is prime
The number is prime
The number is prime
The number is prime
Number is armstrong
120
the number are co prime
32
It is a magic number

# Profiler Report and Debugging

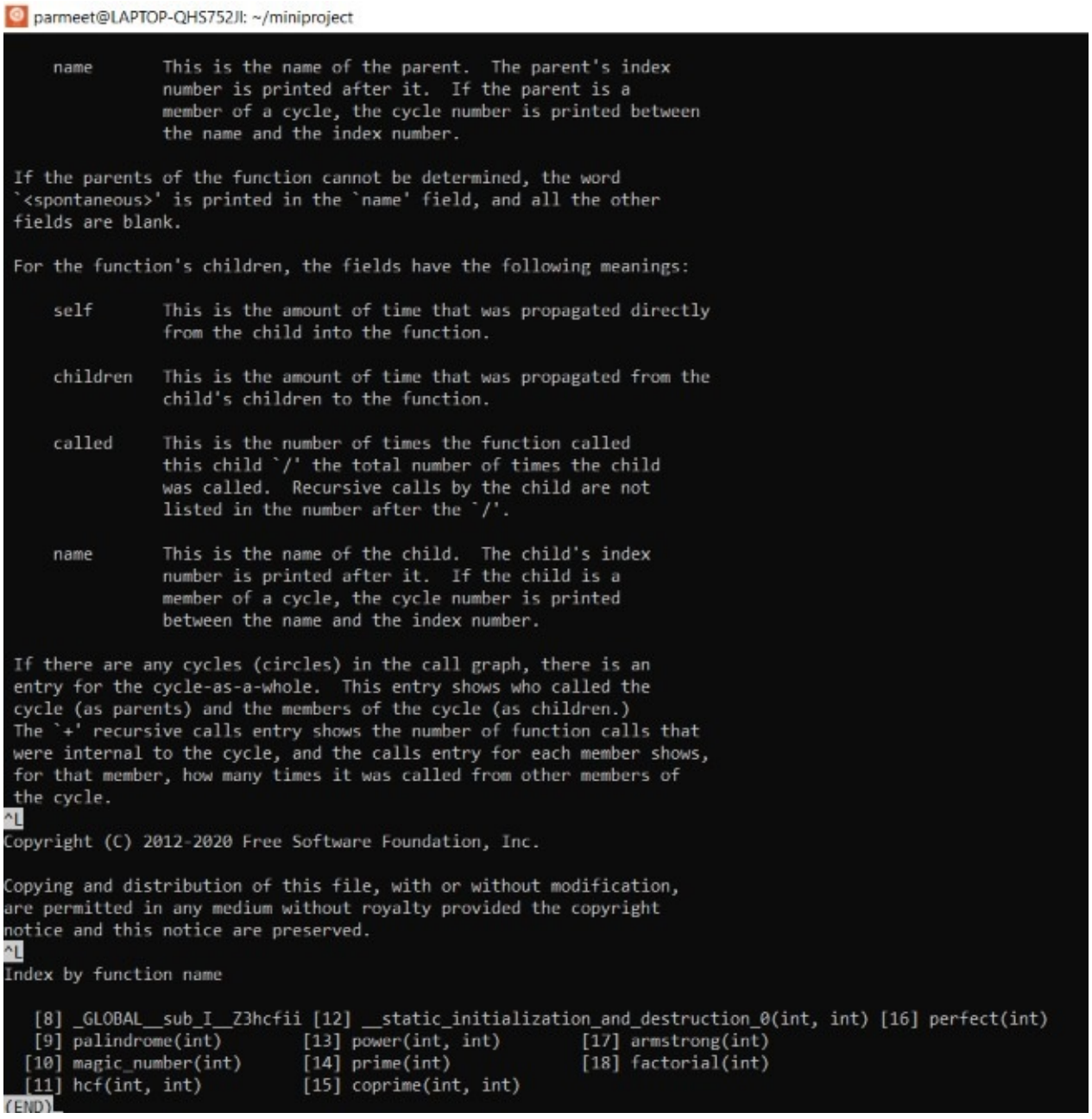

parmeet@LAPTOP-QHS752JI: ~/miniproject
root@LAPTOP-QHS752JI:~# su parmeet
parmeet@LAPTOP-QHS752JI:/root$ cd ..
parmeet@LAPTOP-QHS752JI:/$ cd home
parmeet@LAPTOP-QHS752JI:/home$ cd parmeet
parmeet@LAPTOP-QHS752JI:~$ cd miniproject
parmeet@LAPTOP-QHS752JI:~/miniproject$ g++ -pg -no-pie -fno-builtin functions.cpp -o functions
parmeet@LAPTOP-QHS752JI:~/miniproject$ ls
0 functions functions.cpp gmon.out
parmeet@LAPTOP-QHS752JI:~/miniproject$ ./fxns 10
bash: ./fxns: No such file or directory
parmeet@LAPTOP-QHS752JI:~/miniproject$ ./functions 10
HCF of the numbers is 7
This number is not a palindrome
The number is perfect
The number is prime
The number is prime
The number is prime
The number is prime
The number is prime
Number is armstrong
120
the number are co prime
32
It is a magic number
parmeet@LAPTOP-QHS752JI:~/miniproject$ gprof ./functions | less

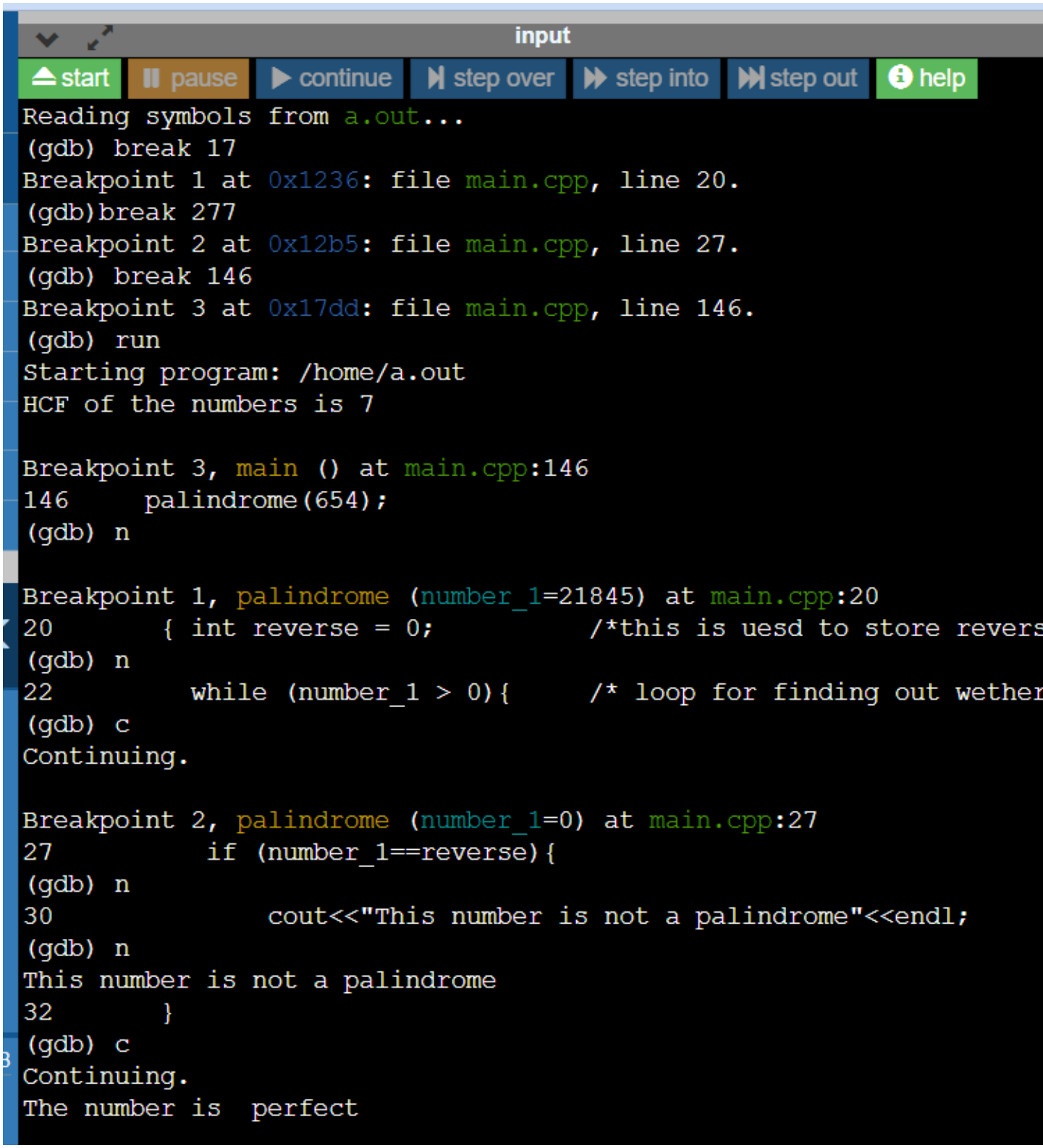

parmeet@LAPTOP-QHS752JI: ~/miniproject
Flat profile:
Each sample counts as 0.01 seconds.
no time accumulated
% cumulative self self total
time seconds seconds calls Ts/call Ts/call name
0.00 0.00 0.00 1 0.00 0.00 _GLOBAL__sub_I__Z3hcfii
0.00 0.00 0.00 1 0.00 0.00 palindrome(int)
0.00 0.00 0.00 1 0.00 0.00 magic_number(int)
0.00 0.00 0.00 1 0.00 0.00 hcf(int, int)
0.00 0.00 0.00 1 0.00 0.00 __static_initialization_and_destruction_0(int, int)
0.00 0.00 0.00 1 0.00 0.00 power(int, int)
0.00 0.00 0.00 1 0.00 0.00 prime(int)
0.00 0.00 0.00 1 0.00 0.00 coprime(int, int)
0.00 0.00 0.00 1 0.00 0.00 perfect(int)
0.00 0.00 0.00 1 0.00 0.00 armstrong(int)
0.00 0.00 0.00 1 0.00 0.00 factorial(int)
% the percentage of the total running time of the
time program used by this function.
cumulative a running sum of the number of seconds accounted
seconds for by this function and those listed above it.
self the number of seconds accounted for by this
seconds function alone. This is the major sort for this
listing.
calls the number of times this function was invoked, if
this function is profiled, else blank.
self the average number of milliseconds spent in this
ms/call function per call, if this function is profiled,
else blank.
total the average number of milliseconds spent in this
ms/call function and its descendents per call, if this
function is profiled, else blank.
name the name of the function. This is the minor sort
for this listing. The index shows the location of
the function in the gprof listing. If the index is
in parenthesis it shows where it would appear in
the gprof listing if it were to be printed.
^L
Copyright (C) 2012-2020 Free Software Foundation, Inc.

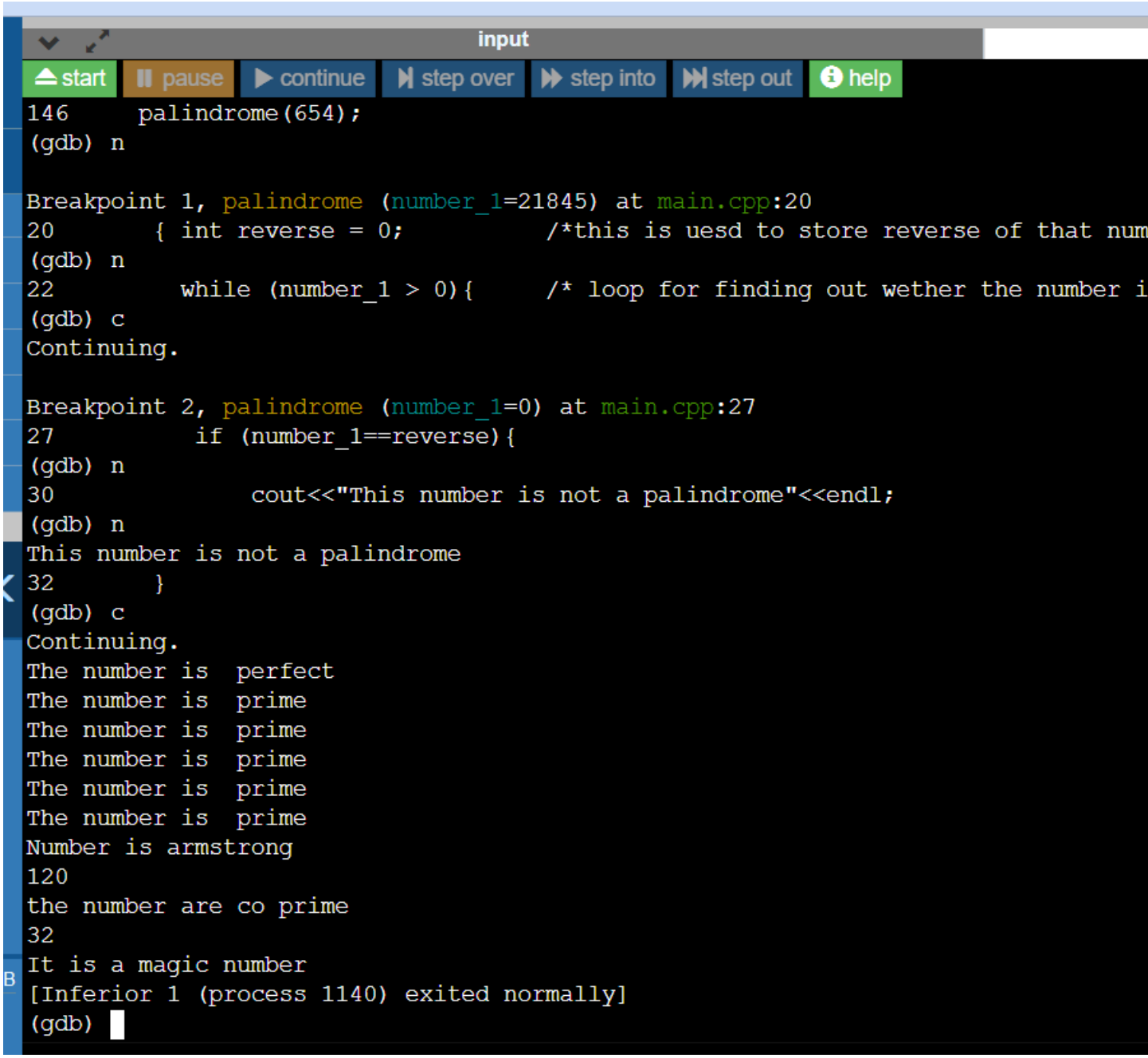

parmeet@LAPTOP-QHS752JI: ~/miniproject
Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
^L
Call graph (explanation follows)
granularity: each sample hit covers 2 byte(s) no time propagated
index % time self children called name
0.00 0.00 1/1 __libc_csu_init [23]
[8] 0.0 0.00 0.00 1 _GLOBAL__sub_I__Z3hcfii [8]
0.00 0.00 1/1 __static_initialization_and_destruction_0(int, int) [12]
0.00 0.00 1/1 main [6]
[9] 0.0 0.00 0.00 1 palindrome(int) [9]
0.00 0.00 1/1 main [6]
[10] 0.0 0.00 0.00 1 magic_number(int) [10]
0.00 0.00 1/1 main [6]
[11] 0.0 0.00 0.00 1 hcf(int, int) [11]
0.00 0.00 1/1 _GLOBAL__sub_I__Z3hcfii [8]
[12] 0.0 0.00 0.00 1 __static_initialization_and_destruction_0(int, int) [12]
0.00 0.00 1/1 main [6]
[13] 0.0 0.00 0.00 1 power(int, int) [13]
0.00 0.00 1/1 main [6]
[14] 0.0 0.00 0.00 1 prime(int) [14]
0.00 0.00 1/1 main [6]
[15] 0.0 0.00 0.00 1 coprime(int, int) [15]
0.00 0.00 1/1 main [6]
[16] 0.0 0.00 0.00 1 perfect(int) [16]
0.00 0.00 1/1 main [6]
[17] 0.0 0.00 0.00 1 armstrong(int) [17]
0.00 0.00 1/1 main [6]
[18] 0.0 0.00 0.00 1 factorial(int) [18]
This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

parmeet@LAPTOP-QHS752JI: ~/miniproject

```
This table describes the call tree of the program, and was sorted by
the total amount of time spent in each function and its children.

Each entry in this table consists of several lines.  The line with the
index number at the left hand margin lists the current function.
The lines above it list the functions that called this function,
and the lines below it list the functions this one called.
This line lists:
    index      A unique number given to each element of the table.
               Index numbers are sorted numerically.
               The index number is printed next to every function name so
               it is easier to look up where the function is in the table.

    % time     This is the percentage of the `total' time that was spent
               in this function and its children.  Note that due to
               different viewpoints, functions excluded by options, etc,
               these numbers will NOT add up to 100%.

    self       This is the total amount of time spent in this function.

    children   This is the total amount of time propagated into this
               function by its children.

    called     This is the number of times the function was called.
               If the function called itself recursively, the number
               only includes non-recursive calls, and is followed by
               a `+' and the number of recursive calls.

    name       The name of the current function.  The index number is
               printed after it.  If the function is a member of a
               cycle, the cycle number is printed between the
               function's name and the index number.


For the function's parents, the fields have the following meanings:

    self       This is the amount of time that was propagated directly
               from the function into this parent.

    children   This is the amount of time that was propagated from
               the function's children into this parent.

    called     This is the number of times this parent called the
               function `/' the total number of times the function
               was called.  Recursive calls to the function are not
               included in the number after the `/'.

.
```


parmeet@LAPTOP-QHS752JI: ~/miniproject
name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.
If the parents of the function cannot be determined, the word `<spontaneous>' is printed in the `name' field, and all the other fields are blank.
For the function's children, the fields have the following meanings:
self This is the amount of time that was propagated directly from the child into the function.
children This is the amount of time that was propagated from the child's children to the function.
called This is the number of times the function called this child `/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the `/'.
name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.
If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The `+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.
^L
Copyright (C) 2012-2020 Free Software Foundation, Inc.
Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.
^L
Index by function name
[8] _GLOBAL__sub_I__Z3hcfii [12] __static_initialization_and_destruction_0(int, int) [16] perfect(int)
[9] palindrome(int) [13] power(int, int) [17] armstrong(int)
[10] magic_number(int) [14] prime(int) [18] factorial(int)
[11] hcf(int, int) [15] coprime(int, int)
(END)


input
start
pause
continue
step over
step into
step out
help
Reading symbols from a.out...
(gdb) break 17
Breakpoint 1 at 0x1236: file main.cpp, line 20.
(gdb)break 277
Breakpoint 2 at 0x12b5: file main.cpp, line 27.
(gdb) break 146
Breakpoint 3 at 0x17dd: file main.cpp, line 146.
(gdb) run
Starting program: /home/a.out
HCF of the numbers is 7
Breakpoint 3, main () at main.cpp:146
146 palindrome(654);
(gdb) n
Breakpoint 1, palindrome (number_1=21845) at main.cpp:20
20 { int reverse = 0; /*this is uesd to store revers
(gdb) n
22 while (number_1 > 0){ /* loop for finding out wether
(gdb) c
Continuing.
Breakpoint 2, palindrome (number_1=0) at main.cpp:27
27 if (number_1==reverse){
(gdb) n
30 cout<<"This number is not a palindrome"<<endl;
(gdb) n
This number is not a palindrome
32 }
(gdb) c
Continuing.
The number is perfect


input
start
pause
continue
step over
step into
step out
help
146 palindrome(654);
(gdb) n
Breakpoint 1, palindrome (number_1=21845) at main.cpp:20
20 { int reverse = 0; /*this is uesd to store reverse of that num
(gdb) n
22 while (number_1 > 0){ /* loop for finding out wether the number i
(gdb) c
Continuing.
Breakpoint 2, palindrome (number_1=0) at main.cpp:27
27 if (number_1==reverse){
(gdb) n
30 cout<<"This number is not a palindrome"<<endl;
(gdb) n
This number is not a palindrome
32 }
(gdb) c
Continuing.
The number is perfect
The number is prime
The number is prime
The number is prime
The number is prime
The number is prime
Number is armstrong
120
the number are co prime
32
It is a magic number
[Inferior 1 (process 1140) exited normally]
(gdb)