Week 6 Homework

Computer Programming Lab

2020/10/21

Remind

- 抄襲一律 0 分 （包含被抄襲者）
- 繳交期限: 10/25(Sun.) 11:59 p.m.
- 繳交的檔案格式、名稱請符合以下規定
  - 請繳交 zip檔至 Ceiba作業區，名稱為 <student_id>.zip
  - 解壓縮後須符合格式、名稱
  - e.g. b12345678.zip
- 必須完成 Demo 才可以提早離開
- 若沒有完成 Demo 就中途早退，視同缺席

Problem 1 - Continued Fraction(1%)

Description

A continued fraction is an expression of the form

$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \dots}}}$$

where a_i and b_i are positive integers. The continued fraction representation for a rational number is finite and only rational numbers have finite representations. In a finite continued fraction (or terminated continued fraction), the iteration/recursion is terminated after finitely many steps by using an integer in replacement of another continued fraction. In contrast, an infinite continued fraction is an infinite expression. In either case, all integers in the sequence, other than the first, must be positive. The integers a_i are called the coefficients or terms of the continued fraction.

Problem 1 - Continued Fraction(1%)

Description

Implement the following two functions in your code:

1. `void rational(int nu, int de)`
    1. `nu` is the numerator
    2. `de` is the denominator
    3. The `rational` function will print the coefficients $a_0, a_1, a_2, \ldots$, each seperated by a whitespace (no whitespace at the end).
2. `void real(long double x, unsigned int n)`
    1. `x` is the input real number
    2. `n` is the number of output numbers
    3. The `real` function will print n coefficients, each seperated by a whitespace (no whitespace at the end).

Problem 1 - Continued Fraction(1%)

Description

```cpp
void rational(unsigned int nu, unsigned int de){/*your code here*/}
void real(double x, unsigned int n){/*your code here*/}
```

C++

All you have to do is finish both the `rational` and the `real` function.

Problem 1 - Continued Fraction(1%)

Input

1. `unsigned int a` and `unsigned int b` for rational number $\frac{a}{b}$
2. a `long double x`, which represents any real number, and a designated `unsigned int n` indicating how many numbers should be printed

Output

1. Coefficients $a_0, a_1, a_2, \ldots$, each seperated by a whitespace (no whitespace at the end)

   For example, since $\frac{415}{93} = 4 + \frac{1}{2+\frac{1}{6+\frac{1}{7}}}$, the output of `rational(415, 93)` should be `4 2 6 7`.

2. n coefficients, each seperated by a whitespace (no whitespace at the end)

   For example, since the value π is roughly $3 + \frac{1}{7+\frac{1}{15+\frac{1}{1+\frac{1}{292}}}}$, the output of `real(3.14159265358979323846, 5)` should be `3 7 15 1 292`.

Problem 1 - Continued Fraction(1%)

Sample input

```
415
93
3.141592653589793238463
5
```

Plain Text

Sample output

```
4 2 6 7
3 7 15 1 292
```

Plain Text

File name

{Student_ID}_1.cpp

Problem 2 - Root Finding(0.5%)

Description

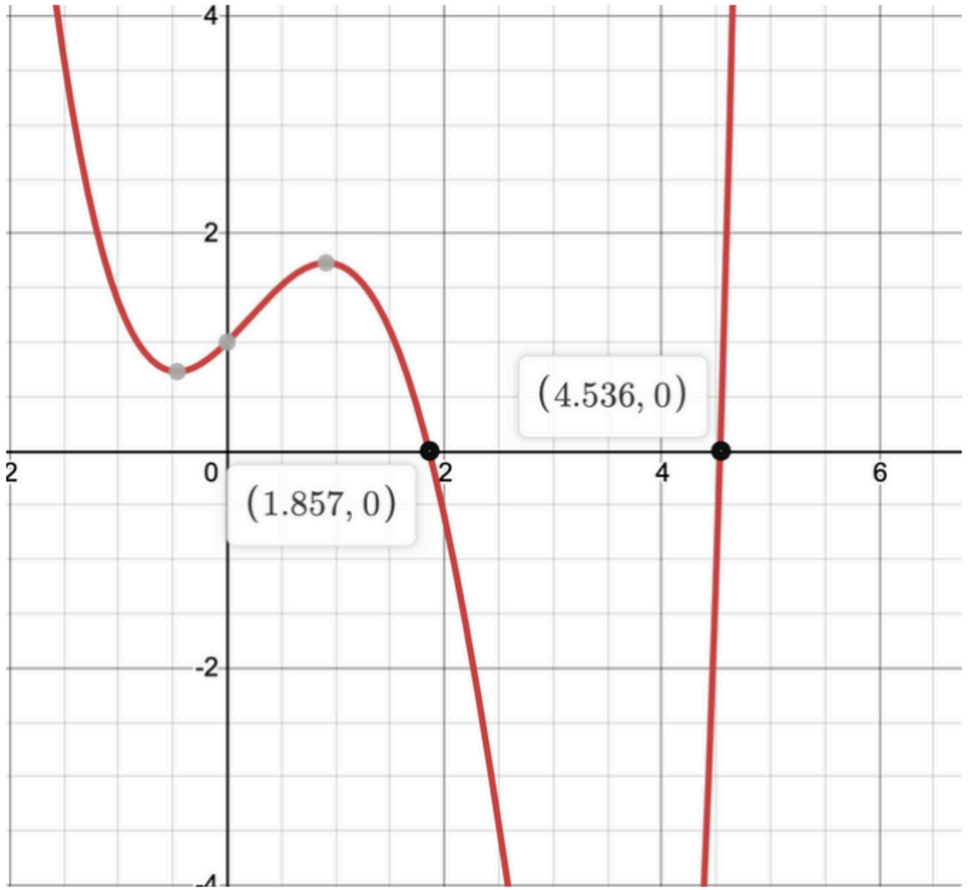
Given a function $f(x)$, we said that an x is a root of f if $f(x) = 0$. For a general equation $h(x) = g(x)$, one can simply rearrange it as $f(x) = h(x) - g(x)$. Then a root of f is a solution to $h(x) = g(x)$. Thus, finding roots is equivalent to solving equations. For equations that can not be solved analytically, i.e., there does not exist closed form solutions, a root finding algorithm is needed. Mainly there are two types of root finding algorithms, one is bracketing algorithms, and the other is non-bracketing algorithms. Both types start with a guess. The main difference between different algorithms is how they make the next guess. A general numerical root finding algorithm is the following:

- Step 1. Make a guess.
- Step 2. Make a new intelligent guess.
- Step 3. Repeat until the required precision is reached.

Problem 2 - Root Finding(0.5%)

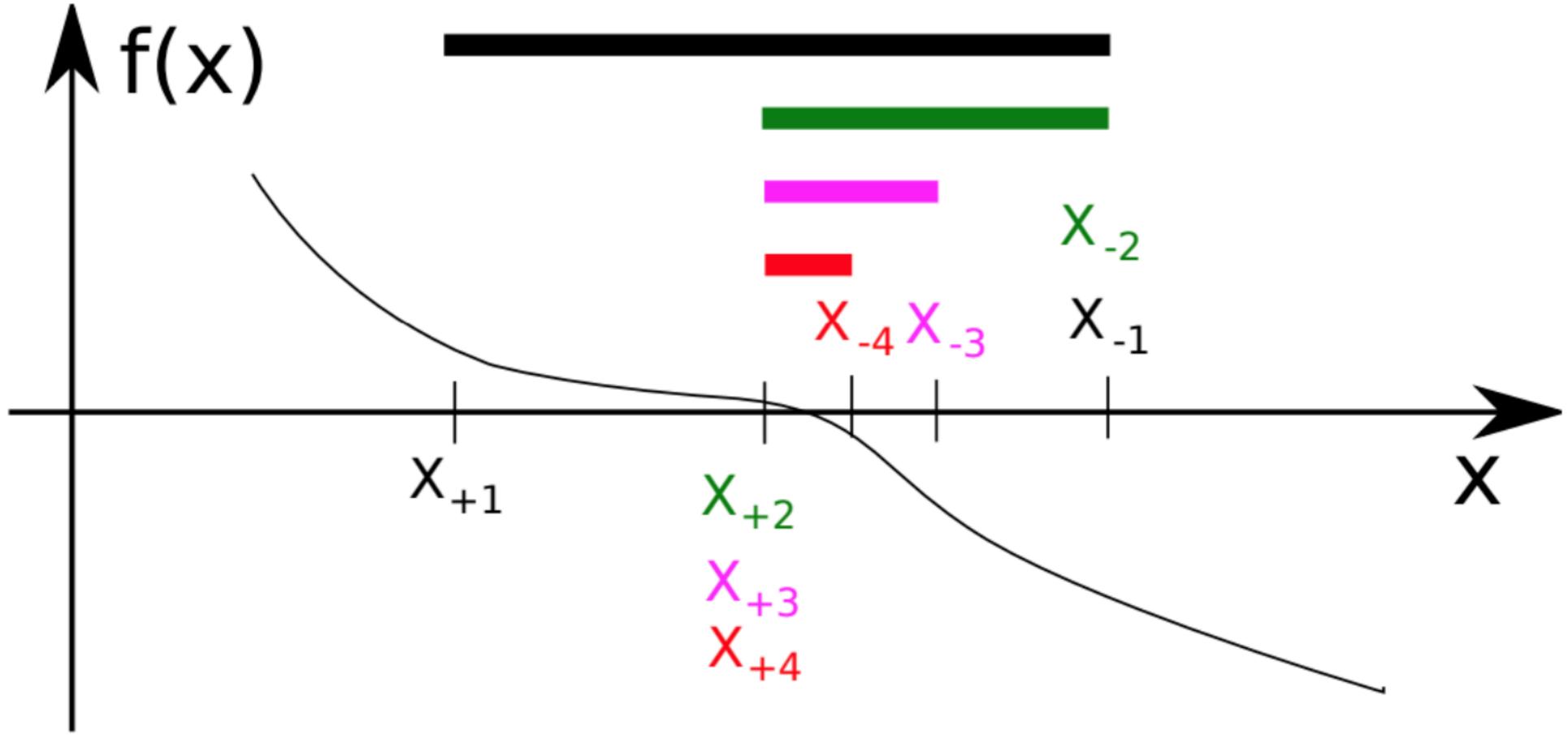
Description

In this problem, we want to solve $e^x = x^3$ using the bisection method and the false position method.



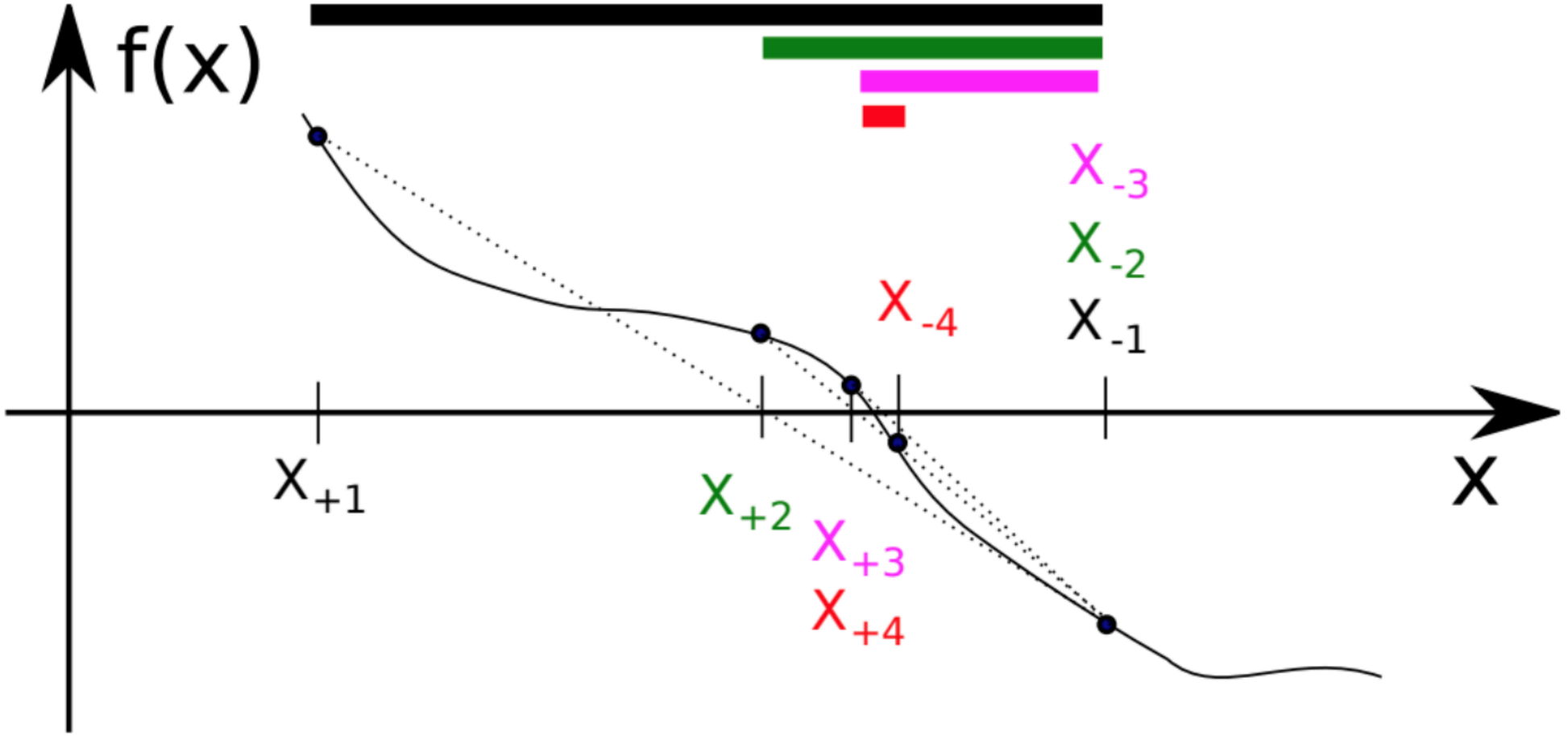
Problem 2 - Root Finding(0.5%)

Bisection method



Problem 2 - Root Finding(0.5%)

False position method



Problem 2 - Root Finding(0.5%)

Description

Implement two functions:

1. `double bisection(double left, double right)`
    1. `left` for the `left` end point of the bracket
    2. `right` for the `right` end point of the bracket
    3. return the appoximated root
2. `double false_position(double left, double right)`
    1. `left` for the `left` end point of the bracket
    2. `right` for the `right` end point of the bracket
    3. return the appoximated root

```cpp
double bisection(double left, double right){/*your code here*/}
double false_position(double left, double right){/*your code here*/}
```

C++

Problem 2 - Root Finding(0.5%)

Input

There is no input for this problem. Use $[0, 4]$ for your initial bracket.

Output

Output two approximated roots in two seperated lines, both having exactly 13 digit after the decimal point.

```cpp
// the approximated root in [0, 4] using the bisection method
// the approximated root in [0, 4] using the false position method
```

C++

File name

{Student_ID}_2.cpp

Problem 3 - Calendar (1.5%)

In the zip file you download from Ceiba, you will find several template files:

All modifications must be done inside the `/*your code here*/` segment, any modification outside the allowed area is strictly forbidden and will result in 0 point.

Note : For problems in 1-1 to 1-4 (4 problems in total), please turn in your code without modifying the filename.

Problem 3-1 Leap Year (revisit) (0.25%)

Description

In "leap.cpp", please implement the function "isLeapYear" to determine whether a year is leap year or not.

The function takes an integer as argument, representing the year , and returns a bool value (T/F) indicating whether the input year is a leap year. More details can be found in the template.

Sample input

```
2020
```

Plain Text

Sample output

```
1
```

Plain Text

File name

leap.cpp

Problem 3-2 Determine Number of Days in a Month(0.25%)

Description

In "num_of_days.cpp", please implement the function "getNumOfDaysInMonth".

The function takes two integer arguments, year and month, and returns another integer, representing how many days the given month in the given year has.

Requirement : You should use the isLeapYear() function you've written for 1-1.

Sample input

```
2020 10
```

Plain Text

Sample output

```
31
```

Plain Text

File name

num_of_days.cpp

Problem 3-3 Determine Starting Day of a Month(0.5%)

Description

In "startday.cpp", please implement the function "getStartDay" to determine what the starting day of a month is. For example, The October of 2020 starts from a Tuesday (2020/10/1).

Again, the function takes two integer arguments, year and month.

It returns an integer, ranging from 0 to 6.

0 represents Sunday, and 1-6 represent Monday to Saturday.

Requirement : Use isLeapYear() & getNumOfDaysInMonth() you've written for 1-1 & 1-2.

Sample input

```
2020 10
```

Plain Text

Sample output

```
4
```

Plain Text

File name

startday.cpp

Problem 3-4 Perpetual Calendar (0.5%)

Description

In "calendar.cpp", you will implement a perpetual calendar (萬年曆)

Below is a diagram for the complete framework, all the functions prototypes are already given inside the "calendar.cpp" template.

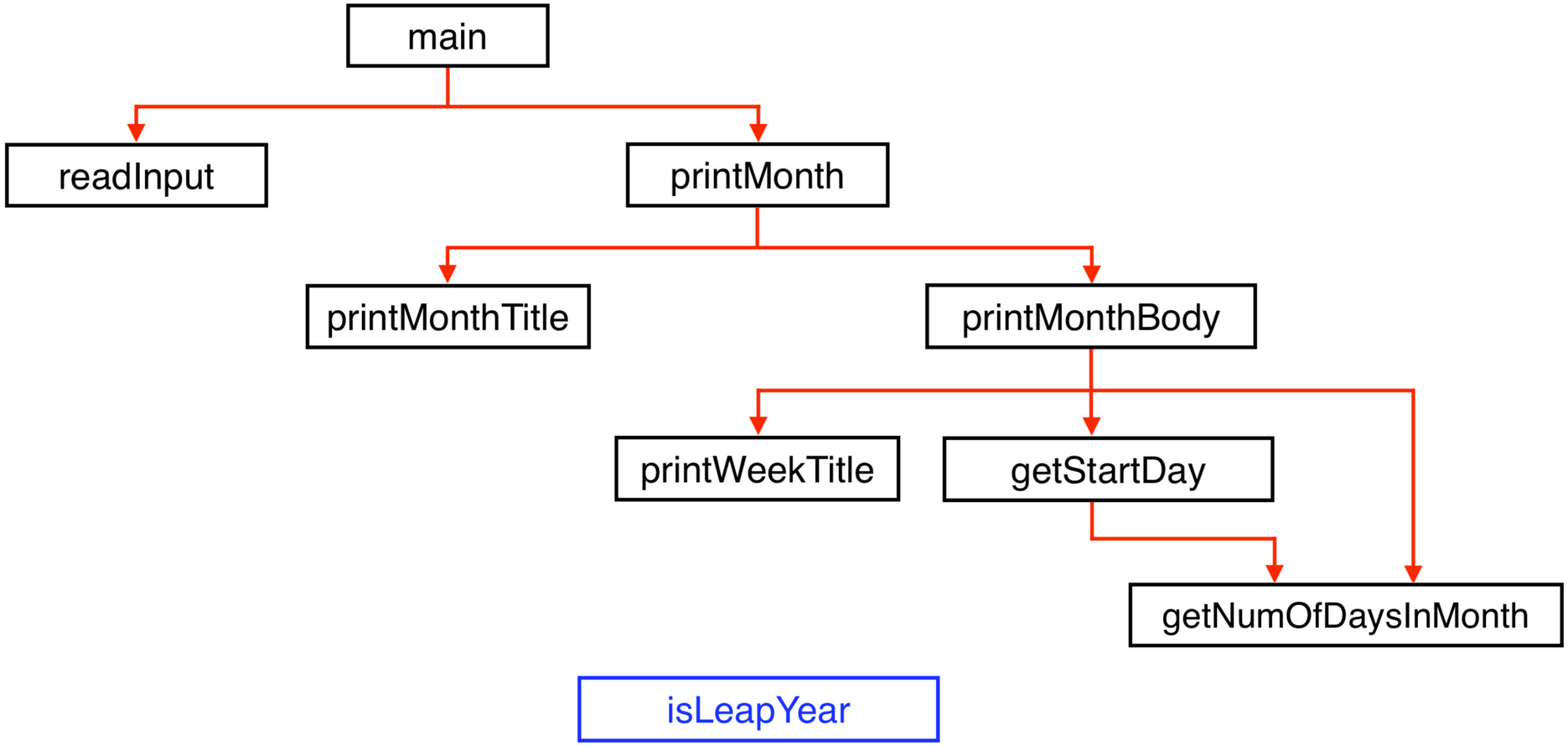
Requirement : You should use all the functions that you implemented in 1-1 to 1-3

Note : As usual, you can only modify the parts in which we have marked as `/*your code here*/`

Any modification outside the allowed area is strictly forbidden and will result in 0 point.

Problem 3-4 Perpetual Calendar (0.5%)

Description



Problem 3-4 Perpetual Calendar (0.5%)

Input

Two numbers which are year and month, respectively. They are both integers, and there is a "/" between them. The year is Common Era (CE), will always be given as anno Domini (AD).

Output

The whole calendar of the given month in the given year.

Problem 3-4 Perpetual Calendar (0.5%)

Sample input

```
2020/10
```

Plain Text

Sample output

```
OCT
SUN MON TUE WED THU FRI SAT
                  1   2   3
  4   5   6   7   8   9  10
 11  12  13  14  15  16  17
 18  19  20  21  22  23  24
 25  26  27  28  29  30  31
```

Plain Text

File name

calendar.cpp