Laboratory 04 – selection statements and loops 02.11.2021

1. [2 points] Write a C program to create a calculator that performs basic arithmetic operations (addition, subtraction, multiplication and division) using `switch/case`. The calculator should take two numbers and the operator from the user and then print the result of the operation according to the entered operator.

a) Declare three variables. Two to hold the values and the third to hold the operator. What types should be variables? Use `scanf` to give them values.
b) The `switch` decides what operation to perform based on the value of the operator.
c) Create 4 `case` labels for 4 arithmetic operations.
d) Use `default` when the user-supplied symbol does not match any of the four arithmetic operations.
e) Print the result on the screen.

Test Data :
Information for the user: `Enter [number 1] [+ - * /] [number 2]`
Data entered by the user: `123 + 34`
Result: `123.00 + 34.00 = 157.00`

Information for the user: `Enter [number 1] [+ - * /] [number 2]`
Data entered by the user: `12 # 34`
Result: `Invalid operator`

2. **[2 points]** Write a C program that finds the maximum of the two numbers entered by the user with a `switch` statement, and check whether the maximum is even or odd using `switch.`

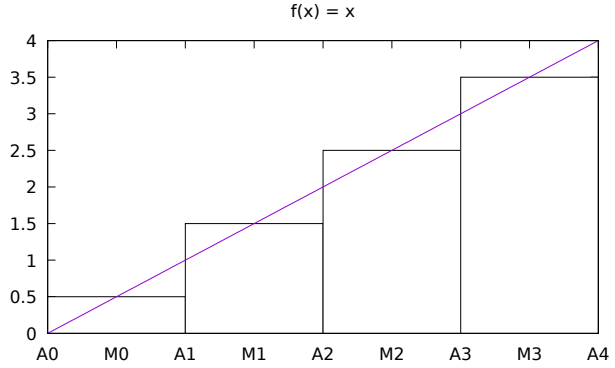
a) Declare three variables. Use `scanf` to initialize two of them.
b) Use `switch` to find maximum. Create two `case` labels, don't use `default`.
c) Print the maximum to the screen.
d) Use `switch` to check whether the maximum is even or odd. Create two `case` labels, don't use `default`.
e) Print info to the screen.

Test Data :
Input: `Enter two numbers: 12 13`
Result:
```
13 is Maximum.
Maximum is Odd.
```

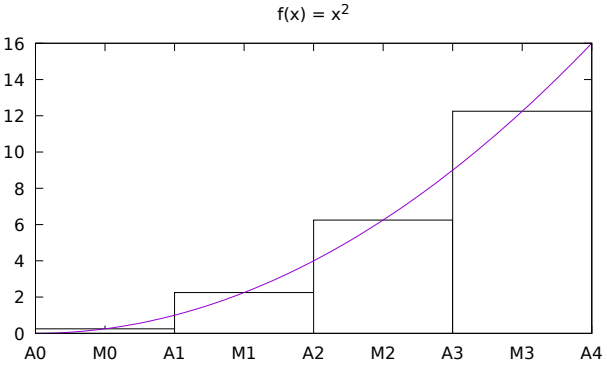
Input: `Enter two numbers: 16 13`
Result:
```
16 is Maximum.
Maximum is Even.
```

3. [**5 points**] Numerical integration comprises a broad family of algorithms for calculating the numerical value of a definite integral. The basic problem in numerical integration is to compute an

approximate solution to a definite integral $\int_a^b f(x)dx$ to a given degree of accuracy. Rectangular integration (a.k.a. The Midpoint Rule) is a numerical integration technique that approximates the integral of a function with a rectangle. It uses rectangles to approximate the area under the curve.



$$\int_{A_0}^{A_4} x\,dx \approx Area\ of\ Rect\ 1 + Area\ of\ Rect\ 2 + Area\ of\ Rect\ 3 + Area\ of\ Rect\ 4$$



$$\int_{A_0}^{A_4} x^2\,dx \approx Area\ of\ Rect\ 1 + Area\ of\ Rect\ 2 + Area\ of\ Rect\ 3 + Area\ of\ Rect\ 4$$

`Area of Rectangle = base * height`
The base of all rectangles is the same length.

`base = A1-A0 = A2-A1 = A3-A2 = A4-A3 =` $\dfrac{A4 - A0}{number\ of\ Rectangles}$

The heights of the rectangles change.
The height of the first rectangle at the point `M0` is `f(M0)`.
We use the midpoint rule because the position of the M0 point is halfway between A0 and A1.
The height of the second rectangle at the point `M1` which is halfway between `A1` and `A2` is `f(M1)`, and so on.

Now just use a loop to calculate the sum of all rectangles to calculate the numerical approximation of the definite integral.

We will calculate the integrals of the following functions. Insert the following code into the program, above the main function.

```
double f1(double x){
  return 1;
}
double f2(double x){
  return x;
}
double f3(double x){
  return x*x;
}
```

a) In the main function, create two variables that will be the beginning and end of the integration interval.
b) Create a variable that will hold the number of rectangles.
c) Create a variable that is equal to the base of the rectangles. Calculate its value using previously defined variables.
d) Create a variable that will hold the result of the integral calculation.

e) Create a loop that iterates through all of the rectangles.
f) In the loop calculate the value of `Mi`, then use it to calculate the height value of the `i-th` rectangle and its area.
g) Add the calculated area to the result.

h) After the loop is complete, print the result of the integral.

Calculate the integral for f1 (x) in the range 0 to 4 for 10 rectangles.
Calculate the integral for f2 (x) in the range 0 to 4 for 10 rectangles.
Calculate the integral for f3 (x) in the range 0 to 4 for 10 rectangles.

Test data:

```
Begin=0.00, End=4.00, Number of recangles=10, base=0.40

result for f1 = 4.00
result for f2 = 8.00
result for f3 = 21.280
```

4. [**3 points**] Let's check how the value of the definite integral depends on the number of rectangles. Add a loop to the previous program that will control the number of rectangles. Here's a function that we're going to integrate in the range 0 to 1.

```
double f(double x){
     return x*x*x*x + x*x*x + x*x + x + 1;
}
```

Perform calculations for 10, 100, 1000 rectangles.

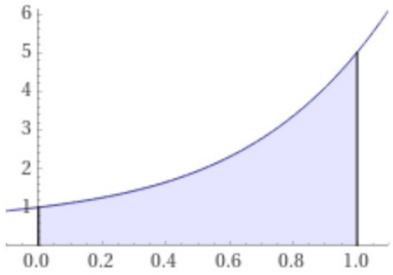
Test data:

```
Begin=0.00, End=1.00

Number of recangles=10, result for f = 2.279586

Number of recangles=100, result for f = 2.283296

Number of recangles=1000, result for f = 2.283333
```

Exact solution from https://www.wolframalpha.com

$$\int_0^1 (x\,x\,x\,x + x\,x\,x + x\,x + x + 1)\,dx = \frac{137}{60} \approx 2.2833$$

Visual representation of the integral



5. [**6 points**] Write a C program that prints all prime numbers in the range `(a,b)`. The program uses two loops and `continue` and `break` statements. Count how many primes are in a given range. Use variable `flagprime` to break the inner loop when we know the number can't be prime. In the inner loop, the `sqrt` function will come in handy.

Program structure:

```
int main(void){

      //declarations and printf

      for(.....){
            char flagprime = .....

            //Special cases 0, 1, 2 and divisible by 2. Use logical operators.
            if (.....)
                  //continue or break

            //Divisibility by odd numbers 3, 5, …
            //What is the loop termination condition?
            for(.....)
                  if(.....){
                        flagprime = .....
                        //continue or break
                  }
            if(flagprime){
                  printf(.....);
                  .....
            }
      }
      printf("\nNumber of primes in (%d, %d): %d\n", .....);

      return 0;
}
```

Test data:

```
Primes in (1, 100):
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

Number of primes in (1, 100): 25
```

6. [**2 points**] Modify the previous program to count how many primes there are in every hundred from 0 to 1000.

Test data:

```
There are 25 primes between 0 and 100.
There are 21 primes between 100 and 200.
There are 16 primes between 200 and 300.
There are 16 primes between 300 and 400.
There are 17 primes between 400 and 500.
There are 14 primes between 500 and 600.
There are 16 primes between 600 and 700.
There are 14 primes between 700 and 800.
There are 15 primes between 800 and 900.
There are 14 primes between 900 and 1000.
```

Next time:
Laboratory 05 – Loops and arrays