Lecture 15
Advanced algorithm with Loops

• The *Rectangle Method*

  • Introduction
  
  ○ Definite integral (*High School* material):
    
    ▪ A definite integral $\int_{a}^{b} f(x) \, dx$ is the integral of a function $f(x)$ with fixed end point $a$ and $b$:
    
    ▪ The integral of a function $f(x)$ is equal to the area under the graph of $f(x)$.
    
    ▪ Graphically explained:

  ○ Rectangle Method:

  ▪ The *rectangle method* (also called the *midpoint rule*) is the *simplest method* in Mathematics used to *compute an approximation* of a definite integral.
1. Divide the **interval** $[a .. b]$ into $n$ pieces; each piece has the **same width**:

$$f(x)$$

The width of each piece of the smaller intervals is equal to:

$$\frac{b - a}{n}$$
2. The **definite integral** (= **area**) under the graph is **approximated** using a **series of rectangles**: 

![Diagram](image)

The **area** of a **rectangle** is equal to:

\[
\text{area of a rectangle} = \text{width} \times \text{height}
\]

We (already) know the **width** of each rectangle:

\[
b - a \\
\text{width} = \frac{b - a}{n}
\]

We still need to find the **height** of each rectangle.
- The **different rectangles** has **different heights**

- Heights of each rectangle:
  
  - The **heights** of each rectangle = the **function value** at the **start** of the (small) **interval**

- Example: first interval

  - First (small) interval: \([a \ldots (a + width)]\) (remember that: \(width = (b - a)/n\))

  - **Height** of first (small) interval:

  ![Diagram](image)

  Therefore: **height** of first rectangle = \(f(a)\)

  - **Area of the rectangle** = \(f(a) \times width\)
Example: second interval

- The second (small) interval is \([a + \text{width}) \ldots (a + 2\text{width})]\) (remember that: \(\text{width} = (b - a)/n\))

- **Height** of first (small) interval:

\[
\text{height} = f(a + \text{width})
\]

Therefore: **height** of first rectangle = \(f(a + \text{width})\)

- **Area of the rectangle** = \(f(a + \text{width}) \times \text{width}\)
We see a pattern emerging:

- Height of rectangle $1 = f(a + 0 \times width)$
- Height of rectangle $2 = f(a + 1 \times width)$
- Height of rectangle $3 = f(a + 2 \times width)$
- ...
- Height of rectangle $n-1 = f(a + (n-2) \times width)$
- Height of rectangle $n = f(a + (n-1) \times width)$

Note: there are $n$ (smaller) interval in total.

Conclusion:

Height of rectangle $i = f( a + (i-1) \times width )$

= the function value at the point "$a + (i-1) \times width$"

$\frac{b-a}{n}$

Recall that: width = -----
- The **width** of rectangle $i$ is equal to:

  \[
  \text{width} = \frac{b - a}{n}
  \]

- The **height** of rectangle $i$ is equal to:

  \[
  \text{height} = f(a + (i-1) \times \text{width})
  \]

- The **area** of rectangle $i$ is equal to:

  \[
  \text{area} = \text{width} \times \text{height} = \text{width} \times f(a + (i-1) \times \text{width})
  \]
The approximation of the definite integral:

\[
\text{Approximation} = \text{sum of the area of the rectangles} \\
= \text{area of rectangle 1} \\
+ \text{area of rectangle 2} \\
+ \ldots \\
+ \text{area of rectangle n} \\
= \text{width} \times f( a + (1-1)\times\text{width} ) \\
+ \text{width} \times f( a + (2-1)\times\text{width} ) \\
+ \ldots \\
+ \text{width} \times f( a + (n-1)\times\text{width} )
\]
Algorithm to compute the sum of the area of the rectangles:

Variables:

```plaintext
double w;       // w contains the width
double sum;     // sum contains the running sum
```

Algorithm to compute the sum:

```plaintext
w = (b - a)/n;  // Compute width
sum = 0.0;      // Clear running sum

for ( i = 1; i <= n; i++ )
{
    sum = sum + w*f( a + (i-1)*w );
}
```
public class RectangleMethod01
{
    public static void main(String[] args)
    {
        double a, b, w, sum, x_i;
        int i, n;

        **** Initialize a, b, n ****

        /* ---------------------------------------------------------
          The Rectangle Rule Algorithm
          --------------------------------------------------------- */
        w = (b-a)/n; // Compute width
        sum = 0.0; // Clear running sum

        for ( i = 1; i <= n; i++ )
        {
            x_i = a + (i-1)*w; // Use x_i to simplify formula...
            sum = sum + ( w * f(x_i) ); // width * height of rectangle i
        }

        System.out.println("Approximate integral value = " + sum);
    }
}
Example 1: compute $\int_0^1 x^3 \, dx$ (the exact answer = 0.25)

```java
public class RectangleMethod01
{
    public static void main(String[] args)
    {
        double a, b, w, sum, x_i;
        int i, n;

        a = 0.0; b = 1.0; // \int_0^1 x^3 \, dx
        n = 1000; // Use larger value for better approximation

        /* ---------------------------------------------------------
         * The Rectangle Rule Algorithm
         * --------------------------------------------------------- */
        w = (b-a)/n;   // Compute width
        sum = 0.0;     // Clear running sum

        for ( i = 1; i <= n; i++ )
        {
            x_i = a + (i-1)*w;
            sum = sum + ( w * (x_i * x_i * x_i) ); // f(x_i) = (x_i)^3
        }

        System.out.println("Approximate integral value = " + sum);
    }
}
```
Nested loop example: parsing

- Problem description:
  - You are given a line of text
  - Example:
    "Hello Mr. Sunshine, How are you ?"
  - Break the line of text into individual words (tokens)
    The words (tokens) are separated by at least 1 space
  - Example output:
    input: "Hello Mr. Sunshine, How are you ?"
    output: "Hello"
      "Mr."
      "Sunshine,"
      "How"
      "are"
      "you"
      "?"
• Reading in a line (with space)
  ○ The following method in the `java.util.Scanner` class will return every character typed until the `RETURN` key is pressed:

```java
nextLine()
```

○ Sample usage:

```java
Scanner input = new Scanner(System.in);

String s;

s = input.nextLine();  // Read a line and store in s
```
A very rudimentary algorithm:

```plaintext
scan_pos = 0;
while ( not done scanning input line )
{
    scan for the next word starting at "scan_pos"
    advance "scan_pos" to the end of the next word
}
```

Example:

```plaintext
0123456789012345678901234  <--- ruler
s = "    abc  Hello  123    "
    ^
    |
scan_pos=0

1a. scan for the next word **starting** at "scan_pos=0"

0123456789012345678901234  <--- ruler
s = "    abc  Hello  123    "
    ^
    |
scan_pos=0

1b. advance "scan_pos" to the end of the next word

0123456789012345678901234  <--- ruler
s = "    abc  Hello  123    "
    ^
    |
scan_pos=6
```
Refining the algorithm

- Input line:

```
0123456789012345678901234
s = "   abc  Hello   123   
```

- We start scanning the input line from the first character:

```
0123456789012345678901234
s = "   abc  Hello   123   
^  
left=0
```
Repeat the following tasks (loop)

- Find the **start** of the **next word**.
  
  How to do it: **Skip** over **leading spaces**:

  ```
  0123456789012345678901234  <--- ruler
  s = "  abc   Hello   123  "
  ^
  |
  left=3
  ```

  **Result**: we found the **start** of a **new word**

- Find the **end** of the **current word**
  
  How to do it: **find** the **first space** (while **holding on** to the **start** of the current word !!!)

  ```
  0123456789012345678901234  <--- ruler
  s = "    abc   Hello   123  "
  ^   ^
  |
  |
  left=3  right=6
  ```

  We must use **another variable** (called: **right**) to find the end of the current word!
- **Yippee!** We found the word.

  We can **extract** the located word with the `substring()` method:

  ```java
  String t;
  t = s.substring(left, right);  // extra word found
  ```

  We are **done** finding one **word**

- **Advance** the **scan position** to the **end of the word found**

  This is achieved with the following statement:

  ```java
  left = right;
  ```

**Explanation:**

**Situation before the statement "left = right;":**

```
 0123456789012345678901234  <-- ruler
s = " abc Hello 123 "
  ^  ^
left=3  right=6
```

**Situation after the statement "left = right;":**

```
 0123456789012345678901234  <-- ruler
s = " abc Hello 123 "
  ^
left=6  ready to scan for the next word !!!
(right=6, but this variable is not relevant)```
Pseudo code:

```
s = input line;
left = 0;
while (left < s.length()) {
    make the "left" index variable to skip over leading space;
    // Result: left points to the start of the next word
    use "right" index variable to find the end of the current word;
    t = s.substring(left, right);  // extract word
    System.out.println(t);        // Print it
    left = right;                // advance scan position to the end of the next word
}
```
There is some **errors** in the above program (because we are not **careful**)

**Example:**

```java
>> java Parse01
Enter line: abc  def  hij  klm
Input line = `abc  def  hij  klm`
Next word found: `abc`
Next word found: `def`
Next word found: `hij`
Next word found: `klm`
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 23
  at java.lang.String.charAt(String.java:558)
  at Parse01.main(Parse01.java:35)
Line 35:
    // Next task: Skip leading spaces
    while ( s.charAt(left) == ' ' )
      left++;
Problem:
    Input line = `abc  def  hij  klm`
      ^
      |  
      left
You will scan PASS the end of the line !!!
```
Re-writing the *for*-statement using a *while*-statement and vice versa

- **While-statement:**

```java
while ( loop-continuation-condition )
{
    statement1
    statement2
    ....
}
```

*Body of the while-loop*

This can be written as follows using a *for*-statement:

```java
for ( ; loop-continuation-condition ; )
{
    statement1
    statement2
    ....
}
```
For-statement:

```
for (init-expression; loop-cont-condition; incr-expression)
{
    statement1
    statement2
    ....
}
```

*Body of the for-loop*

that *does not contain* a *continue-statement*, can be *written* as follows using a *while-statement*:

```
init-expression ;
while ( loop-cont-condition )
{
    statement1
    statement2
    ....
    incr-expression ;
}
```
If a *for*-statement does not contain any *continue* statements, you can **re-write** the *for*-statement as follows:

```
for (init-expression; loop-cont-condition; incr-expression)
{
    statement1
    statement2
    ....
}
```

**Re-write**

```
init-expression;
for (  ; loop-cont-condition ;  )
{
    statement1
    statement2
    ....
    incr-expression ;
}
```