**ITI1120 – Introduction to Computing I**

**Exercise Workbook**

**Fall 2013**

Table of Contents

[1. Introduction 3](#_Toc313795577)

[2. Computer Programming Model 3](#_Toc313795578)

[3. Pre-defined Algorithm Models 6](#_Toc313795579)

[4. Summary of Program Structure, Algorithm Concepts and Java Programming Concepts 7](#_Toc313795580)

[5. Section 1 Exercises 8](#_Toc313795581)

[6. Section 2 Exercises 12](#_Toc313795582)

[7. Section 3 Exercises 14](#_Toc313795583)

[8. Section 4 Exercises 17](#_Toc313795584)

[9. Section 5 Exercises 26](#_Toc313795585)

[10. Section 6 Exercises 35](#_Toc313795586)

[11. Section 7 Exercises 55](#_Toc313795587)

[12. Section 8 Exercises 63](#_Toc313795588)

[13. Section 9 Exercises 73](#_Toc313795589)

[14. Section 10 Exercises 83](#_Toc313795590)

[15. Section 11 Exercises 87](#_Toc313795591)

# Introduction

This work book contains the exercises that shall be studied in class. Please bring it to all classes. The front matter (sections 1 to 4) of the workbook provides some background information that can be useful in completing the exercises as well as help in completing assignments. It includes:

* **Computer Programming Model** – The programming model is a representation of the main computer components that are used to execute programs. The model is useful in understanding how a program is executed. It is used with most of the exercises in the work book. The model is briefly described in Section 2. This section also gives two blank programming model pages that can be used for your studying and assignments.
* **Pre-defined Algorithm Models** – A number of pre-defined “standard” algorithm models are available for developing algorithms. They represent standard functions available in a computer system such as reading from the keyboard and writing to the terminal console window.
* **Program Structure, Algorithm and Java Programming Concepts** – This section provides a table that summarizes many of the programming structure and concepts studied in class and during the labs. The table can serve as a quick reference for the important representation of algorithm concepts and Java programming concepts/details.

# Computer Programming Model

The two main components used in executing a computer program are the computer’s main memory and the central processing unit (CPU). The computer memory contains the program in the form of machine instructions – these machine instructions are typically created from higher level languages such as C, Java (using compilers – more on this in class).

The machine instructions operate on variables, which are also located in the computers memory. Essentially the CPU reads values from and writes values to the variables (i.e the locations in memory where variables are located). Operations such as addition, incrementing, testing, are all typically done within the CPU.

Computer memory consist of a sequence of bytes (8 bit values; a bit is a digit that can assume the value 0 or 1 and a byte contains 8 bits). Thus a computer that contains 3 GigaBytes of memory contains 3 x 1024 x 1024 x 1024 bytes, that is, over 3 billion bytes.

Each of these bytes has an address. An address is simply a number that points to each byte in memory. Thus when the CPU wants to execute an instruction it reads the instruction using its address (the CPU actually keeps track of the address of instructions to execute them in sequence). Instructions contain the addresses of the variables to manipulate them.

Memory is divided into regions – part of the memory will contain the program to execute (the machine instructions), part of the memory contains variables used for executing the sub-programs of a program (such memory is retrieved after the execution of a sub-program and re-used during the execution of other sub-programs), and part of the memory is used to store variables and other data structures that are available during the whole execution of a program (for example, for storing arrays and objects). The program model represents each of these memory parts as the **program memory**, the **working memory**, and the **global memory** respectively. Note that the working memory is divided into pieces to illustrate how working memory is “reserved” for the execution of different sub-programs.

The following two pages provide blank working models so that you may use them for studying and completing assignments. The first page does not include global memory and can be used during most of the first half of the course since global memory is used when arrays are introduced.

**Program Memory Working memory**

CPU

**Program Memory Working memory Global Memory**

CPU

# Pre-defined Algorithm Models

The following algorithm models can be used for developing algorithms. Most of the algorithms represent input and output for interacting with the user. The last three algorithms represent other tasks supported by the computer (or language).

printLine(<argument list>)

Prints on the terminal screen the contents of the argument list. The argument list consists of a comma separated list of items that can be a string (e.g. “ a string”) or a variable name. After the output, the cursor on the terminal is placed on the next line.

print(<argument list>)

Prints on the terminal screen the contents of the argument list. The argument list consists of a comma separated list of items that can be a string (e.g. “ a string”) or a variable name. After the output, the cursor remains after the last character printed.

intVar ← readInteger()

Reads an integer number from the keyboard

numVar ← readReal()

Reads a real number from the keyboard

charVar ← readCharacter()

Reads a single character from the keyboard.

boolVar ← readBoolean()

Reads “true” or “false” and stores TRUE or FALSE; it gives a Boolean value as a result.

(intArray, n) ← readIntLine

Reads a line of input, extracts the integers from the line and adds them into a integer array (note that intArray is a reference to the array). The variable n represents the number of elements in the array.

(realArray, n) ← readRealLine

Reads a line of input, extracts the real values from the line and adds them into a real array (note that realArray is a reference to the array). The variable n represents the number of elements in the array.

(charArray, n) ← readCharLine

Reads a line of input, and adds the characters on the line into a character array (note that charArray is a reference to the array). The variable n represents the number of elements in the array.

strVar ← readLine()

Reads a line of input, i.e. a string from the keyboard. strVar is a reference to a string.

numVar ← random():

Generates a random value (result numVar) greater or equal to 0.0 and less than 1.0.

anArrayRefVar ← makeNewArray( num )

Creates an array, referenced by anArrayRefVar, which has num positions with unknown values in them.

aMatrixRefVar ← makeNewMatrix(numR, numC )

Creates a matrix, referenced by aMatrixRefVar, which has numR rows and numC columns of elements with unknown values in them.

The following conventions shall be used in both algorithm models and Java programs:

* All variable names shall start with a lower case character (examples: num, x, arrayRef).
* All algorithm/method names shall start with a lower case character (examples: get, set, getValue).
* When a name is composed of two or more words, capitals are used to start each word except the first word of the name (examples: aMatrixRefVar, readReal).
* Class names shall start with an upper case character (examples: Student, Course).
* If you use constants, use all upper case characters (PI, CONST\_VAL). The underscore can be useful to separate words.

# Summary of Program Structure, Algorithm Concepts and Java Programming Concepts

| **Algorithm Model** | **Java** | | |
| --- | --- | --- | --- |
| **Instruction Block** | | **Primitive Types**  **int** integers  **double** real values  **char** single characters  **boolean** Boolean values |
|  | {  *<instruction>*  *<instruction>*  *.*  *.*  } |
| **Subprogram** | | **Arrays / Matrices**  // Array  ***Type*** [] ***varname***;  ***varname***= new***Type***[***size***];  // Matrix  ***Type*** [][] ***varname***;  ***varname*** = new ***Type***[***#Rows***][***#Cols***];  // ***Type*** – any variable type (int,  // double, etc.) or class name  // ***varname*** – legal variable name  // ***size*** – number of elements  // ***#Rows*** – number of rows  // ***#Cols*** – number of columns |
| …  Result ← Name(*<paramList>*) | public static *<type>* name(*<paramList>*)  {  *<instruction>*  *<instruction>*  *.*  *.*  } |
| **Instructions** | |
| Operations | |
| ( ) *(expression)* [ ] *(array subscript)* . *(object member)*  + – *(positive / negative valeu)* NOT *(logical operator)*  \* / MOD *(multiply, divide, and modulo)*  + - *(addition or subtraction of two values, concatenation)*  < > ≥ ≤ *(comparison)*  = ≠ *(comparison)*  AND *(logical operator)*  OR *(logical operator)*  🡠 *(assignment variable)* | ( ) *(expression)* [ ] *(array subscript)* . *(object member)*  + – *(to indicate positive or negative values)* ! *(not)*  \* / % *(multiply, divide, and modulo)*  + - *(for addition or subtraction of two values, concatenation)*  < > >= <= *(comparison)*  == != *(comparison)*  && *(logical AND)*  || *(logical OR)*  = *(assignment to variable)* |
| Simple Expression | | **Classes / objects**  class ***ClassName***  {  // Attributes  // Constructor  public ***ClassName***(***parameterList***)  {  }  // Methods  .  .  .  }  // ***ClassName*** – valid name  // keywords used in  // declaring attributes  // and methods  public – accessible by all  private – accessible by class only  static – class variable or method  // Reference variable  ***ClassName varname*;**  // Creating an object  ***varname*** = new ***ClassName***(); |
| z ← x + y | z = x + y; |
| Calls to subprograms | |
| average ← markResults(score1,  score2, score3) | average = markResults(score1,  score2, score3); |
| Branch Instruction | |
|  | if (*<logical expression>*)  {  *<instruction>*  *<instruction>*  *.*  *.*  }  else  {  *<instruction>*  *<instruction>*  *.*  *.*  } |
| Pre-Test Loop Instruction | | Available Class/Methods  ITI1120 Methods:  ITI1120.readInt( )  ITI1120.readDouble( )  ITI1120.readChar( )  ITI1120.readBoolean( )  ITI1120.readDoubleLine( )  ITI1120.readIntLine( )  ITI1120.readCharLine( )  ITI1120.readString( )  Scanner Methods:  Scanner keyboard =  new Scanner(System.in);  keyboard.nextInt( )  keyboard.nextDouble( )  keyboard.nextBoolean( )  keyboard.nextLine( )  System.out Methods:  System.out.println( … )  System.out.print( … )  Math Class Methods/Attributes  Math.sin(rad) //cos, tan, asin,  // acos, PI  Math.exp(x) // also log(x)  Math.log(x) // also log10(x)  Math.pow(x, a) // x to power a  Math.sqrt(x)  Math.random()  Math.abs(x) |
|  | while(*<logical expression>*)  {  *<instruction>*  *<instruction>*  *.*  *.*  } |
| Post-Test Loop Instruction | |
|  | do  {  *<instruction>*  *<instruction>*  *.*  *.*  }  while(*<logical expression>*); |
| Alternative Loop Instruction (Java **for** loop) | |
|  | for(*<expressionInit>;*  *<logical expression>;*  *<expressionUpdate>*)  {  *<instruction>*  *<instruction>*  *.*  *.*  } |

# Section 1 Exercises

**Program Memory Exercise 1-1 - Algorithm for Average Working memory**

**GIVENS: num1, num2, num3 (three numbers)**

**RESULTS:**

**avg (the average of num1, num2, and num3)**

**HEADER:**

**BODY:**

CPU

**Program Memory** **Exercise 1-2 - Another example**  **Working memory**

GIVENS:

RESULTS:

HEADER:

BODY:

GIVENS:

**INTERMEDIATES**:

RESULTS:

HEADER:

BODY:

CPU

**Program Memory Exercise 1-3 Average out of 100 Working memory**

Givens:

Results:

Intermediates:

Header:

Body:

CPU

**Program Memory Exercise 1-4 Last Example Working memory**

… Without a constant

…

Body

…

tax1 🡨 c1 \* 0.07 *//* GST

tax2 🡨 c2 \* 0.07 *//* GST

tax3 🡨 c3 \* 0.07 *//* PST

tax4 🡨 c4 \* 0.07 *//* GST

…

…with constants GST and PST

CPU

# Section 2 Exercises

**Exercise 2-1 – Collating Sequence**

**'A' < 'a'** is

while

**'?' < ' '** is

**Exercise 2-2 - Test for Upper case**

* Suppose the variable **x** contains a value of type **char**.
* Write a Boolean expression that is TRUE if the value of **x** is an upper case letter and is FALSE otherwise.
  + Note that you don't need to know the actual code value of the characters!

**Exercise 2-3 - Operator Precedence**

* What is the order of evaluation in the following expressions?

**a + b + c + d + e**

**a + b \* c - d / e**

**a / (b + c) - d % e**

**a / (b \* (c + (d - e)))**

**Program Memory Exercise 2-4 - Some math Working memory**

**// Variable declarations**

**// Compute the average**

**// Variable declarations**

**//Compute the quotient and remainder**

CPU

# Section 3 Exercises

score1

score2

score3

sum

avgOutOf25

avgPct

**markResult**

**main**

**Program Memory Exercise 3-1 – Main Algorithm Working memory**

Givens:

Results:

Intermediates:

Header: **main()**

Body:

Givens: **score1, score2, score3** (scores out of 25)

Results: **avgPct** (average of scores, out of 100)

Intermediates: **sum** (sum of scores)

**avgOutOf25** (average of scores, out of 25)

Header: **avgPct ← markResult( score1, score2, score3 )**

Body:

**1. sum ← score1 + score2 + score3**

**2. avgOutOf25 ← sum / 3**

**3. avgPct ← avgOutOf25 \* 4**

CPU

**Program Memory Exercise 3-2 – Swap 2 Values Working memory**

GIVENS:

RESULTS:

MODIFIEDS:

INTERMEDIATES:

HEADER

BODY:

CPU

**Program Memory Exercise 3-3 – Translating to Java Working memory**

**public static void main (String[] args)**

**{**

**// SET UP KEYBOARD INPUT**

**Scanner keyboard = new Scanner( System.in );**

**// DECLARE VARIABLES/DATA DICTIONARY**

**// READ IN Values from the user**

**// Call to markResults**

**// PRINT OUT RESULTS**

**}**

**public static double markResult(double score1,**

**double score2,**

**double score3 )**

**{**

**// Intermediate variables**

**// Result variable**

**// BODY OF ALGORITHM**

**// RETURN RESULTS**

**}**

score1

score2

score3

sum

avgOutOf25

avgPct

first

second

third

average

**markResult**

**main**

CPU

# Section 4 Exercises

score1

score2

score3

sum

avgOutOf25

avgPct

**MarkResult**

**Program Memory Exercise 4-1 Tracing Example Working memory**

Call: avgPct ← markResult(18, 23, 19)

Givens: score1, score2, score3 (scores out of 25)

Results: avgPct (average of scores, out of 100)

Intermediates:

sum (sum of scores)

avgOutOf25 (average of scores, out of 25)

Header: avgPct ← markResult( score1, score2, score3 )

Body:

1. sum ← score1 + score2 + score3

2. avgOutOf25 ← sum / 3

3. avgPct ← avgOutOf25 \* 4

CPU

Table 1: Trace for avgPct ← markResult(18, 23, 19)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Statement | score1 | score2 | score3 | sum | avgOutOf25 | avgPct |
| Initial values |  |  |  |  |  |  |
| 1. sum ← score1 + score2 + score3 |  |  |  |  |  |  |
| 2. avgOutOf25 ← sum / 3 |  |  |  |  |  |  |
| 3. avgPct ← avgOutOf25 \* 4 |  |  |  |  |  |  |

**Program Memory Exercise 4-2 Tracing a Call Working memory**

Givens: none

Results: none

Intermediates:

**first, second, third** (three scores)

**average** (average of scores, out of 100)

Header: **main()**

Body:

(Read in scores from the user)

1. **printLine(“Please enter three scores”)**
2. **first ← readReal()**
3. **second ← readReal ()**
4. **third ← readReal ()**

(Call the markUser algorithm)

1. **average ← markResult(first, second, third)**

(Print the average for the user)

1. **printLine(“The average is “, average)**

Givens: **score1, score2, score3** (scores out of 25)

Results: **avgPct** (average of scores, out of 100)

Intermediates: **sum** (sum of scores)

**avgOutOf25** (average of scores, out of 25)

Header: **avgPct ← markResult( score1, score2, score3 )**

Body:

**1. sum ← score1 + score2 + score3**

**2. avgOutOf25 ← sum / 3**

**3. avgPct ← avgOutOf25 \* 4**

**main**

first

second

third

average

**markResult**

score1

score2

score3

sum

avgOutOf25

avgPct

**Table 1**: Trace for main algorithm:

CPU

Terminal Window

Interaction with user:

Please enter three scores out of 25

***23 16 21***

The average is 80 percent

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Statements | first | second | third | average |
| Initial values |  |  |  |  |
| 1. printLine(“Please enter three scores”) |  |  |  |  |
| 2. first ← readReal() |  |  |  |  |
| 3. second ← readReal () |  |  |  |  |
| 4. third ← readReal () |  |  |  |  |
| 5. Call average ← markResult(first, second, third) (See Table 2) |  |  |  |  |
| 6. printLine(“The average is “, average) |  |  |  |  |

Call algorithm MarkResult:

average ← markResult (first, second, third)

avgPct ← markResult (score1, score2, score3)

**Table 2** – Trace for avgPct ← markResult (23. 16. 21)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Statement | score1 | score2 | score3 | sum | avgOutOf25 | avgPct |
| Initial values |  |  |  |  |  |  |
| 1. sum ← score1 + score2 + score3 |  |  |  |  |  |  |
| 2. avgOutOf25 ← sum / 3 |  |  |  |  |  |  |
| 3. avgPct ← avgOutOf25 \* 4 |  |  |  |  |  |  |

**Program Memory Exercise 4-3 – Marks out of 100, again Working memory**

Givens:

Results:

Intermediates:

Header:

Body:

GIVENS: num1, num2, num3 (three numbers)

RESULTS: avg(the average of num1, num2, and num3)

HEADER: avg 🡠 average(num1, num2, num3)

BODY:

1. avg 🡠 (num1 + num2 + num3)/3

num1

num2

num3

avg

CPU

**Program Memory Exercise 4-4 – Tracing Example Working memory**

score1

score2

score3

avgOutOf25

avgPct

Call to average ← markResult( 23, 16, 21 )

Givens: score1, score2, score3 (scores out of 25)

Results: avgPct (average of scores, out of 100)

Intermediates:

avgOutOf25 (average of scores, out of 25)

Header: avgPct ← markResult(score1, score2, score3 )

Body:

1. avgOutOf25 ← average(score1, score2, score3)

2. avgPct ← avgOutOf25 \* 4

GIVENS: num1, num2, num3 (three numbers)

RESULTS: avg (the average of num1, num2, and num3)

HEADER: avg 🡠 average(num1, num2, num3)

BODY:

1. avg 🡠 (num1 + num2 + num3)/3

num1

num2

num3

avg

**Exercise 4-4 – Tracing Example**

CPU

**Table 1** - Table for avgPct ← markResult(23, 16, 21)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Statements |  |  |  |  |  |
| Initial values |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Call algorithm Average:

**Table 2** - Table for avg ← average( )

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Statement | num1 | num2 | num3 | sum | avg |
| Initial values |  |  |  |  |  |
| 1. sum ← num1 + num2 + num3 |  |  |  |  |  |
| 2. avg ← sum / 3 |  |  |  |  |  |

**Program Memory Exercise 4-5 – Reverse Digits Working memory**

GIVENS:

RESULTS:

IINTERMEDIATES:

HEADER:

BODY:

The following algorithm is available to extract the ten’s and one’s digits from a two digit number:

(high, low) ← digits( x )

CPU

**Program Memory Exercise 4-6 Trace Reverse Digits Working memory**

Table1 - Trace for reverseN 🡠 rev2(42))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Statements | n | tens | ones | reverseN |
| Initial values |  |  |  |  |
| 1. Call digits(N) | |  |  |  |  |
| 2. reverseN ← ones \* 10 + tens | |  |  |  |  |

Call to (tens, ones) ← digits(n)

(tens, ones) ← digits(n)

(high, low) ← digits(x)

**Program Memory Exercise 4-7 – Join Four Numbers Working memory**

Givens:

Result:

Intermediates:

Header:

Body:

You may assume there is available an algorithm:

c ← join( a, b )

Givens: a, b, two positive integers

Result: c is the number having the digits

in a followed by the digits in b.

CPU

# Section 5 Exercises

**Program Memory Exercise 5-1 - Back to the Larger of Two Numbers Working memory**

**GIVENS:**

**RESULT:**

**HEADER:**

**BODY:**

CPU

**Program Memory Exercise 5-2 - Maximum of 3 numbers Working memory**

**GIVENS:**

**RESULT:**

**HEADER:**

**BODY:**

**(Nested tests)**

**OR**

**(Sequence of tests)**

CPU

**Program Memory Exercise 5-3 – Translating Branches 1 Working memory**

**GIVENS: x, y, z (three numbers)**

**RESULT: m (the larger of x, y and z)**

**HEADER: m ← max3( x, y, z)**

**BODY:**

**Sequence of tests**

**public double max3(double x,**

**double y, double z)**

**{**

**double m;**

**return m;**

**}**

CPU

**Program Memory Exercise 5-4 - Translation of Nested Branches Working memory**

**GIVENS: x, y, z (three numbers)**

**RESULT: m (the larger of x, y and z)**

**HEADER: m ← max3( x, y, z)**

**BODY:**

**Nested tests**

**public double max3(double x,**

**double y, double z)**

**{**

**double m;**

**return m**

**}**

CPU

**Program Memory Exercise 5-5 – Translation of an Instruction Bloc Working memory**

**{**

**}**

**false**



a



4

b



0

**true**

**false**

**true**

x > 0 ?

y > x ?

a



5

b



a + 1

a



0

b



0

CPU

**Program Memory Exercise 5-6 - Trace of MAX3, version 2**  **Working memory**

**GIVENS: x, y, z (three numbers)**

**RESULT: m (the larger of x, y and z)**

**HEADER: m ← max3(x, y, z)**

**BODY:**

m



y

Ø

y > m ?

false

true

m



z

Ø

z > m ?

false

true

m



x



1

2

3

4

5

6

7

CPU

x

y

z

m

Table 1 - Trace: MAX3(5, 11, 8)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x | y | z | m |
| Initial values |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Program Memory Exercise 5-7 - Movie Tickets**  **Working memory**

GIVENS: age (persons age)

RESULT: cost (ticket cost)

HEADER: cost ← ticketCost(age)

BODY:

(Version 1: Nested Tests)

(Version 2: Sequence of Tests)

CPU

**Program Memory Exercise 5-8 - Positive Value**  **Working memory**

GIVEN:

RESULT:

HEADER:

BODY:

CPU

**Exercise 5-9 Compound Boolean Expressions**

**Exercise 5-10 - More Compound Boolean Expressions**

Suppose x = 5 and y = 10.

|  |  |
| --- | --- |
| Expression | Value |
| (x > 0) AND (NOT (y = 0)) |  |
| (x > 0) AND ((x < y) OR (y = 0)) |  |
| (NOT (x > 0)) OR ((x < y) AND (y = 0)) |  |
| NOT ((x > 0) OR ((x < y) AND (y = 0))) |  |

# Section 6 Exercises

**Program Memory Exercise 6-1 - Sum from 1 to N**  **Working memory**

**GIVEN:**

**INTERMEDIATE:**

**RESULT:**

**HEADER:**

**BODY:**

CPU

**Exercise 6-1: Trace of sum1toN(3)**

**Working Memory**

|  |  |  |  |
| --- | --- | --- | --- |
| Instructions | n | count | sum  n  sum  count |
| *Init.* |  |  |  |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. TRUE |  |  |  |
| 4. |  |  |  |
| 5. |  |  |  |
| 3. TRUE |  |  |  |
| 4. |  |  |  |
| 5. |  |  |  |
| 3. TRUE |  |  |  |
| 4. |  |  |  |
| 5. |  |  |  |
| 3. FALSE |  |  |  |

**Program Memory Exercise 6-1 - Sum from 1 to N: Variation (I)**  **Working memory**

**GIVEN:**

**INTERMEDIATE:**

**RESULT:**

**HEADER:**

**BODY:**

CPU

**Program Memory Exercise 6-1 - Sum from 1 to N: Variation (II)**  **Working memory**

**GIVEN:**

**INTERMEDIATE:**

**RESULT:**

**HEADER:**

**BODY:**

CPU

**Program Memory Exercise 6-2 – A definite loop Working memory**

**GIVENS:**

**RESULTS:**

**INTERMEDIATES:**

**ASSUMPTIONS:**

**HEADER:**

**BODY:**



CPU

**Program Memory Exercise 6-3 – An indefinite loop Working memory**

**GIVENS:**

**RESULTS:**

**INTERMEDIATES:**

**ASSUMPTIONS:**

**HEADER:**

**BODY:**

CPU

**Program Memory Exercise 6-4 – Translate Loop to Java Working memory**

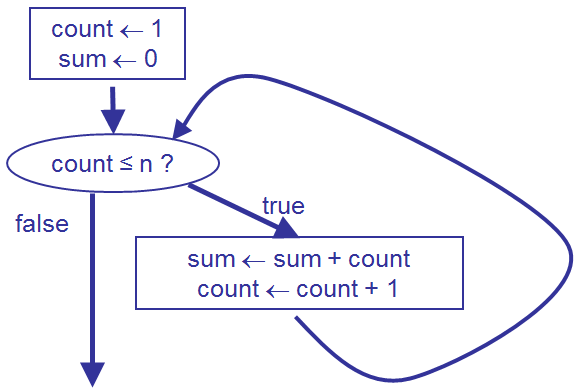
**GIVEN: n (a positive integer)**

**INTERMEDIATE: count (index going from 1 to N)**

**RESULT: sum (sum of integers 1 to N)**

**HEADER: sum ← sum1ToN(n)**

**BODY:**



CPU

**Program Memory Exercise 6-5 – Example of Post-Test Loop**  **Working memory**

**GIVENS:**

**RESULTS:**

**INTERMEDIATES:**

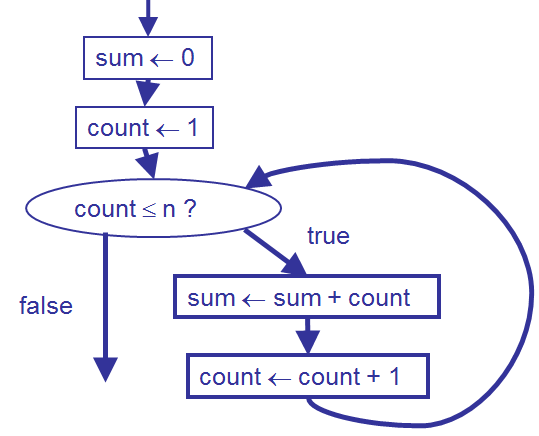
**CONSTRAINTS:**

**HEADER:**

**BODY:**

CPU

**Program Memory Exercise 6-6 – FOR loop to add 1 to N**  **Working memory**

****

**Exercise 6-7 – Array Indexing**

CPU

* The index (subscript) of an array of length l may be any integer expression that returns a value in the range 0...(l-1).
  + Suppose k =2, and A references

**2**

**-1**

**5**

**7**

(length 4)

a[2] =

a[k] =

a[2\*k-1] =

a[a[0]+1 ] =

**Program Memory Exercise 6-8 – Value in Middle of Array Working memory Global Memory**

GIVENS: a (an array of real numbers)

n (an odd integer, the length of a)

RESULT: mid (the middle member of a)

INTERMEDIATES:

HEADER:

BODY

CPU

**Program Memory Exercise 6-9 – Swap Values in an Array Working memory Global Memory**

GIVENS: a (an array of integers)

i, j (two indices)

MODIFIEDS:

INTERMEDIATES:

RESULT:

HEADER: swap( a, i, j )

BODY

**Program Memory Exercise 6-10 – Creating an Array Working memory Global Memory**

CPU

GIVENS: n (a positive integer)

RESULTS: a (an reference to an array

containing n…1)

INTERMEDIATES:

HEADER:

BODY:

Trace for n=3

|  |  |  |  |
| --- | --- | --- | --- |
| statements |  |  |  |
| initial values |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

| **Algorithm Model** | **Java** |
| --- | --- |
| **Exercise 6-11** - Find the sum of the values in an array containing N values | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  INTERMEDIATE:  RESULT: sum (Sum of array contents)  HEADER: sum ← sumArray(a,n)  BODY: | public static int[] sumArray(int[] a)  {  } |
| **Exercise 6-12 (a)** - Given a value t and an array x containing n values, check if the sum of x’s values exceeds t. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  t (A “threshold” value)  INTERMEDIATE:  RESULT: exceeds (Boolean: True if sum > t  and False otherwise)  HEADER: exceeds ← sumLargerThanT(a,n,t)  BODY: | public static boolean sumLargerThanT(int[] a, int t)  {  } |
| **Exercise 6-12 (b)** - Given a value t and an array x containing n values, check if the sum of x’s values exceeds t. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  t (A “threshold” value)  INTERMEDIATE:  RESULT: exceeds (Boolean: True if sum > t  and False otherwise)  HEADER: exceeds ← sumLargerThanT(a,n,t)  BODY: | public static boolean sumLargerThanT(int[] a, int t)  {  } |
| **Exercise 6-13** - Count how many times k occurs in an array containing n values. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  k (Value for which to count instances)  INTERMEDIATES:  RESULT: count (Number of times value of k is  contained in A)  HEADER: count ← countK(a,n,k)  BODY: | public static int countK(int[] a, int k)  {  // Results  } |
| **Exercise 6-14 (a)** - Given an array x of n values and a number k, see if k occurs in x or not. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  k (Value to find)  INTERMEDIATES:  RESULT: found (Boolean : true if k is in array   and false otherwise)  HEADER: found ← findK(a,n,k)  BODY: | public static boolean findK(int[] a, int k)  {  } |
| **Exercise 6-14 (b)** - Given an array x of n values and a number k, see if k occurs in x or not. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  k (Value to find)  INTERMEDIATES:    RESULT: found (Boolean : true if k is in array   and false otherwise)  HEADER: found ← findK(a,n,k)  BODY: | public static boolean findK(int[] a, int k)  {    } |
| **Exercise 6-15** - Given an array x of n values and a number k, find the position of the first occurrence of k. (If k does not occur, return –1 as the position.) | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  k (Value to find)  INTERMEDIATES:  RESULT: position (Position of k in array, or  -1 if k is not contained in array)  HEADER: position ← whereIsK(a,n,k)  BODY: | public static int whereIsK(int a[], int n, int k)  {  }  } |
| **Exercise 6-16** - Find the maximum value in an array containing n values. | |
| GIVENS: n (a positive integer)  a (array containing n values)  INTERMEDIATE:  RESULT: max (maximum member of a)  HEADER: max ← maxInArray( a, n )  BODY | public static int maxInArray(int [] a)  {  } |
| **Exercise 6-17 (a)** - Find the position of the first occurrence of the maximum value in an array containing n values. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  INTERMEDIATES:  RESULTS: position (Position of Maximum value contained in a)  HEADER: position ← maxPosInArray(a,n)  BODY: | public static int maxPosInArray(int [] a)  {  } |
| **Exercise 6-17 (b)** - Find the position of the first occurrence of the maximum value in an array containing N values. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  INTERMEDIATES:  RESULTS: position (Position of Maximum  value contained in a)  HEADER: position ← maxPosInArray(a,n)  BODY: | public static int maxPosInArray(int [] a)  {  } |
| **Exercise 6-17 (c)** - Find the position of the first occurrence of the maximum value in an array containing n values. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  INTERMEDIATES:  RESULTS: position (Position of Maximum  value contained in a)  HEADER: position ← maxPosInArray(a,n)  BODY: | public static int maxPosInArray(int [] a)  {  } |
| **Exercise 6-18** - Check if an array of n values contains any duplicates. | |
| GIVENS: a (An array of numbers)  n (Number of array elements)  INTERMEDIATES:  RESULT: duplicates (True if there are duplicates in   a and false otherwise)  HEADER: duplicates ← hasDuplicates(a,n)  BODY: | public static boolean hasDuplicates(int [] a)  {  } |

**Exercise 6-19**: Comparing Strings

* What is the value of **result** for these examples?
  + Example 1:

**String str1 = "abcde";**

**String str2 = "abcfg";**

**int result = str1.compareTo(str2);**

* + Example 2:

**String str1 = "abcde";**

**String str2 = "ab";**

**int result = str1.compareTo(str2);**

# Section 7 Exercises

**Program Memory Exercise 7-1 – Swap Values in an Array Working memory Global Memory**

**class SwapTilYouDrop**

**{**

**public static void main (String args[ ])**

**{ int i = 0;**

**int a;**

**a = new int[]{ 2, 4, 6, 8, 10, 12 };**

**while( i <= 2 )**

**{**

**arraySwap(a, i, 5 - i ) ;**

**i = i + 1;**

**}**

**for ( i = 0 ; i <= 5 ; i = i + 1 )**

**{ System.out.println("a[" + i + "] is "**

**+ a[i] ); }**

**}**

**}**

**// arraySwap : swaps values at i,j**

**// Givens: x, an array,**

**// i,j, 2 indices in x**

**public static void arraySwap(int[ ] x,**

**int i,int j)**

**{**

**// DECLARE VARIABLES**

**int temp ; // Inter: holds x[i]**

**// BODY OF ALGORITHM**

**temp = x[i] ;**

**x[i] = x[j] ;**

**x[j] = temp;**

**}**

**}**

2nd call

1st call

3rd call

Exercise 7-1: Trace – Table 1, page 1, main

Terminal Window

CPU

**?**

**a**

Output

**?**

**?**

Initial values

**array**

**i**

Statement

Exercise 7-1: Trace – Table 1, page 2, main

Exercise 7-1: Trace – Table 2, arraySwap(a, 0, 5)

(most recent values from page 1)

**a**

Output

**Array**

**i**

Statement

**arraySwap( a, i, 5-i)**

**arraySwap( x, i, j )**

**temp**

**j**

**i**

Initial values

**Array in Table 1**

**x**

Statement

**arraySwap( a, i, 5-i)**

**arraySwap( x, i, j )**

**temp**

**j**

**i**

Initial values

**Array in Table 1**

**x**

Statement

Exercise 7-1: Trace – Table 3, arraySwap(a, 1, 4)

Exercise 7-1: Trace – Table 4, arraySwap(a, 2, 3)

**arraySwap( a, i, 5-i)**

**arraySwap( x, i, j )**

**temp**

**j**

**i**

Initial values

**Array in Table 1**

**x**

Statement

Exercise 7-2 : Validating Numbers

main

readIntLine

isZero

valid

readDigits

sumDigits

**ValidCardNumber.java**

**DigitsLib.java**

**ITI1120.java**

| **Algorithm Model** | **Java** |
| --- | --- |
| **ValidCardNumber Class (ValidCardNumber.java)** | |
| GIVENS: (none)  RESULT: (none)  INTERMEDIATE:  digits (reference to an integer array of digits)  n (number of elements in the array)  testValid (Indicated if card is valid)  HEADER: main  BODY: | */\* the main method calls the other methods in order to solve*  *particular tasks. \*/*  public static void main (String [ ] args)  {  int [ ] digits; // reference variable to array of digits  boolean testValid; // indicates if card is valid  // body  digits = DigitsLib.readDigits( ); *// call to read the data*  while ((digits.length == 4) &&  (!DigitsLib.isZero(digits)))  {  *// sends the number to the valid() method*  testValid = valid(digits);  *// print the result*  if (testValid)  { System.out.println("This number is valid."); }  else  { System.out.println("This number is invalid."); }  }  } |
| GIVENS: digits (reference to array of digits)  n (number of elements in the array)  RESULT: isValid (TRUE if card number is valid, FALSE otherwise)  ITERMEDIATE:  firstThree (first three digits of last group)  lastDigit (last digit of credit card number)  sum (sum of 15 digits)  HEADER: isValid 🡠 valid(digits, n)  BODY: | */\* This method validates the credit card number \*/*  private static boolean valid(int [ ] digits)  {  int firstThree; // first three digits of last group  int lastDigit; // last digit of credit card number  int sum; // sum of first 15 digits  boolean isValid; // result: true if number is valid  *// find the first 3 digits pf the last group*  firstThree =  *// find the last digit of the number*  lastDigit =  *// find the sum of the first 15 digits*  sum =  *// determines the validity*  isValid **=**  return isValid;  } |
| **DigitsLib Class (DigitsLib.java)** | |
| GIVENS: digits (reference to array of digits)  n (number of elements in the array)  RESULT: flag (TRUE if first digit is zero, FALSE otherwise)  INTERMEDIATE: (none)  HEADER: flag 🡠 isZero(digits, n)  BODY: | *// first version: only the first digits need to be 0*  public static boolean isZero(int [ ] digits)  {  boolean flag; // result  flag = digits[0] == 0;  return(flag);  }  *//second version: all 16 digits need to be 0*  public static boolean isZero(int [ ] digits)  {  } |
| GIVENS: (none)  ITERMEDIATE:  RESULT: intArray (reference to integer array)  n (number of elements in the array  HEADER: intArray, n 🡠 readDigits( )  BODY: | */\* This method asks the user to input a credit card number as 4 integers, that will be placed in an array. This method calls readIntLine( ) from the class ITI1120 to read the array in integers. \*/*  public static int [ ] readDigits( )  {  int [ ] intArray; // reference to array  System.out.println  ("Please input the credit card number as four ");  System.out.println  ("numbers of four digits, separated by spaces;");  System.out.println("or press 0 to finish.");  intArray = ITI1120.readIntLine( );  } |
| GIVENS: x (integer number)  RESULT: sum (sum of all digits in the integer number)  INTERMEDIATE:  HEADER: sum 🡠 numDigits(x)  BODY: | *// Returns the sum of the digits of a number x*  public static int sumDigits(int x)  {  int sum; // result – sum of digits  // Body  sum = 0;  while (x != 0)  {  }  return sum;  } |

# Section 8 Exercises

**Program Memory Exercise 8-1 –** Trace for this value of X **Working memory Global Memory**

x

n

partialS

s

Trace call recSum(X, 3)

GIVENS: x (an array of integers)

n (number of elements to sum in x)

RESULT: x (sum of n elements in x)

INTERMEDIATES:

partialS (partial sum of first m elements in x)

HEADER:

s 🡨 recSum(x, n)



x

n

partialS

s

Exercise 8-1 – Trace, Table 1 – recSum(x, 3)

x

n

partialS

s

CPU

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| statement  X  **3**  **5**  **2** | Ref. by x | n | partialS | s |
| initial values |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

partialS ← recSum(x,n-1)

s ← recSum (x,n)

Exercise 8-1 – Trace, Table 2 – recSum(x, 2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| statement | Ref. by x | n | partialS | s |
| initial values |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

partialS ← recSum(x,n-1)

s ← recSum (x,n)

Exercise 8-1 – Trace, Table 3 – recSum(x, 1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| statement | Ref. by x | n | partialS | s |
| initial values |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Program Memory Exercise 8-2 – Translating SumRec to Java Working memory Global Memory**

GIVENS: X (an array of integers)

N (number of elements to sum in A)

RESULT: S (sum of N elements in A)

INTERMEDIATES:

PartialS (partial sum of first M elements in A)

HEADER:

S 🡨 RecSum(X, N)



CPU

| **Algorithm Model** | **Java** |
| --- | --- |
| **Exercise 8-3 (a)** - Find xn where x and n are integers and n ≥ 0, x ≥1. (Version 1) | |
| GIVENS: x (base of exponentiation)  n (power to which x is raised)  RESULT: xToTheN (x to the power of n)  INTERMEDIATES:  m (set to n-1; smaller!)  xToTheM (partial results)  HEADER: xToTheN ← power (x,n)  BODY: | *// METHOD power: find x to the power n*  public static int power(int x, int n)  {  *// VARIABLE DECLARATION / DATA DICTIONNARY*  int m; *// INTERMEDIATE: reduced value*  int xToTheM; *// INTERMEDIATE: partial result*  int xToTheN; *// RESULT: expected result*  *// ALGORITHM BODY*  *// RETURN RESULT*  return xToTheN;  } |
| **Exercise 8-3 (b)** - Find xn where x and n are integers and n ≥ 0, x ≥1. (Version 2 – efficient version) | |
| GIVENS: x (base of exponentiation)  n (power to which x is raised)  RESULT: xToTheN (x to the power of n)  INTERMEDIATES:  m (set to n-1; smaller!)  xToTheM (partial results)  HEADER: xToTheN ← power (x,n)  BODY: | *// METHODE power: find x to the power n*  public static int power(int x, int n)  {  *// VARIABLE DECLARATION / DATA DICTIONNARY*  int m; *// INTERMEDIATE: reduced value*  int xToTheM; *// INTERMEDIATE: partial result*  int xToTheN; *// RESULT: expected result*  *// ALGORITHM BODY*  *// RETURN RESULT*  return xToTheN;  } |
| **Exercise 8-4** - Given an array a of more than n numbers, return TRUE if all the numbers in positions 0…n of a are equal, and false otherwise.. | |
| GIVENS: a (An array of numbers)  n (Number of array elements to test)  RESULT: isEqual (Boolean: TRUE if all values in  elements up to position n are equal)  INTERMEDIATE:  partialRes (partial result)  HEADER: isEqual ← upToNSame(a, n)  BODY:  Efficient Version | // Method upToNSame – Are elements up to n in a equal?  public static boolean upToNSame(int[] a, int n)  {  *// VARIABLE DECLARATION / DATA DICTIONNARY*  boolean partialRes; *// INTERMEDIATE: partial result*  boolean isEqual; *// RESULT: expected result*  *// ALGORITHM BODY*  *// RETURN RESULT*  return isEqual;  }  // Method upToNSame – efficient version  public static boolean upToNSame(int[] a, int n)  {  *// VARIABLE DECLARATION / DATA DICTIONNARY*  boolean isEqual; *// RESULT: expected result*  *// ALGORITHM BODY*  *// RETURN RESULT*  return isEqual;  } |
| **Exercise 8-5** - Calculate n !. | |
| Givens: n *(integer)*  Results: f *(integer, n factorial)*  Intermediates:  Header: f 🡨 fact(n)  Body: | *// Method fact*  *// Given: n, an integer*  public static int fact(int n)  {  } |
| **Exercise 8-6** - Find the sum of 1+2+…+n. | |
| GIVENS: n (An integer)  RESULT: sumN (sum of integers from 1 to n)  INTERMEDIATE:  m (set to n-1; smaller!)  sumM (sum of integers from 1 to m)  HEADER: sumN ← sumToN(n)  BODY: | public static int sumToN(int n)  {  // Variable Declarations  int sumN; // RESULT  int m; // INTERMEDIATE  int sumM; // INTERMEDIATE  // Return results  return(sumN);  } |
| **Exercise 8-7** - Given an array a of n characters, reverse the values stored in positions Start to Finish. | |
| GIVENS: a (reference to a char. array to sort)  low (low index )  high (high index )  RESULTS: (none)  MODIFIED: A (references sorted array)  INTERMEDIATES:  newHigh (new high index)  newLow (new low index)  temp (used for swapping)  HEADER reverse(a, low, high)  BODY: | *// Method : reverse: Reverse the characters in array a with // size n.*  *// To be called initially with reverse(a, 0, n-1)*  public static void reverse(char [] a, int low, int high)  {  *// VARIABLE DECLARATION / DATA DICTIONNARY*  int newHigh; *// INTERMEDIATE: smaller high*  int newLow; *// INTERMEDIATE: greater low*  char temp; *// INTERMIDATE: buffer for char*  *// ALGORITHM BODY*  *// RESULT*  } |
| **Exercise 8-8** - Sort an array of numbers in increasing order: – sort algorithm/method | |
| GIVENS: a (reference to an array to sort)  n (number of elements in array)  RESULTS: (none)  MODIFIED: a (sorted array)  INTERMEDIATES:  maxPos (position of largest value in array)  temp (used for swapping)  HEADER sort( a, n )  BODY: | public static void sort(int[] a, int n)  {  // VARIABLE DECLARATIONS  // GIVENS: a – reference to array to sort  // n – number of elements to sort – note  // that a.length CANNOT be used.  // INTERMEDIATES  int maxPos; // position of largest value  int temp; // used for swapping  // BODY  } |
| **Exercise 8-8** - Sort an array of numbers in increasing order: – locateLargestValue algorithm/method | |
| GIVENS: a (an array to sort)  n (number of elements in array)  RESULTS: pos (index of largest value in array)  INTERMEDIATES:  m (integer, smaller interval)  posM (position of largest value in  smaller array)  HEADER pos ← LocateLargestValue( a, n ) | public static void locateLargestValue(int[] a, int n)  {  // VARIABLE DECLARATIONS  // GIVENS: a – reference to array to sort  // n – number of elements to sort – note  // that a.length CANNOT be used.  int pos; // RESULT – position of largest value  // INTERMEDIATES  int m; // smaller  int posM; // used for swapping  // RESULT  return(pos);  } |

# Section 9 Exercises

* Exercise 9-1: The matrix m is an array of 4 arrays, each with 6 members. If m is regarded as a 2-dimensional array, then

m[1][2] is

m[2][5] is

m[4][1] is

m[3] is

**Program Memory Exercise 9-2 – Max value in a matrix Working memory Global Memory**

m

nRows

nCols

row

col

max

GIVENS: m (a matrix)

nRows (the number of rows in m)

nCols (the number of columns in m)

INTERMEDIATES:

row (index of current row)

col (index of current column)

RESULT: max (the maximum value in the

matrix)

HEADER: max ← findMatrixMax(m, nRows, nCols)

BODY:

Alternative:

Adr1

**Program Memory Exercise 9-3 – Diagonal-check algorithm Working mem Global Mem**

.

.

.

CPU

GIVENS: m (a matrix)

nRows (the number of rows in A)

nCols (the number of columns in A)

INTERMEDIATES:

row (index of current row)

col (index of current column)

RESULT: isDiagonal (TRUE if matrix is diagonal, false otherwise)

HEADER: isDiagonal ← checkDiag( m, nRows, nCols )

BODY:

Efficient Version

CPU

**Program Memory Exercise 9-4 – Max value in a matrix in Java Working memory Global Memory**

m

nRows

nCols

row

col

max

Adr1

**// Note: Integer.MIN\_VALUE is the most**

**// negative allowable integer for a Java**

**// int, and can be used for –∞.**

**public static int matrixMax (int[][] m,**

**int nRows, int nCols)**

**{**

**max = Integer.MIN\_VALUE;**

**// m[0][0] is an alternate choice**

**int max; // if the matrix m is not empty**

**int row; // INTERMEDIATE**

**int col; // INTERMEDIATE**

**// Body**

**return max;**

**}**

**Program Memory Exercise 9-5 – Reading a Matrix Working memory Global Memory**

## CPU

public static int readMatrix(int nRows, int nCols)

{

int row, col;

int nRows, nCols;

int[][] m ;

}

.

.

.

m

nRows

nCols

row

col

Adr1

Adr4

Adr3

Adr2

.

.

.

.

.

.

**Program Memory Exercise 9-6 –** Find Cheap Direct Flights **Working memory Global Memory**

.

.

.

CPU

cost

home

d

n

aCity

numCities

cities

Adr1

GIVENS:

home (the number of the city you live)

cost (reference to the cost matrix)

d (the amount you afford)

n (the total number of cities)

RESULTS:

cities (reference to an array of cities which

you can visit)

INTERMEDIATE:

aCity (the city we are currently checking)

numCities (the number of cities to which you can go)

HEADER

(numCities, cities) ← cheapDirectFlights (home, cost, d)

.

.

.

Adr4

Adr3

Adr2

.

.

.

.

.

.

AdrX

.

.

.

**Program Memory Exercise 9-6 (cont.) –** Find Cheap Direct Flights in Java **Working memory Global Memory**

.

.

.

CPU

cost

home

d

n

aCity

numCities

tempCities

cities

public static int[] cheapDirectFlights(int home,

int[][] cost, int d, int n )

{

int[] cities; // RESULT: an array of cities we

// can afford to visit

// INTERMEDIATES:

int aCity ; // The city currently checking

int numCities ; // Number of cities we can visit

int[] tempCities; // Temporary array for cities

// BODY

// Now return the array of cities with correct length

return cities;

}

Adr1

.

.

.

Adr4

Adr3

Adr2

.

.

.

.

.

.

AdrX

.

.

.

AdrY

**Program Memory Exercise 9-7–** Delete a Row  **Working mem Global Mem**

.

.

.

CPU

GIVENS: M (a square matrix)

N (number of rows and columns in M)

R (row number to be removed)

RESULTS: (none)

MODIFIEDS: M (the original matrix with row R

removed, and all rows moved up by one)

INTERMEDIATES: Index (index of row being moved)

HEADER: DeleteRow(M, N, R)

BODY:

GIVENS**:** m *(a square matrix)*

n *(size of M)*

r *(number of row to move, R > 0)*

RESULT**:** (none)

MODIFIED**:** m *(row r copied to row r-1)*

INTERMEDIATE**:** index *(column index)*

HEADER: moveUp(m, n, r)

BODY:

public static void deleteRow(int [ ][ ]m,

int n, int r)

{

int index; *// INTERMEDIATE*

}

private static void moveUp(int [ ][ ]m,

int n, int r)

{

int index; *// INTERMEDIATE*

}

**Program Memory Exercise 9-7 –** Delete a Row  **Working mem Global Mem**

CPU

GIVENS: m (a square matrix)

n (number of rows and columns in m)

r (row number to be removed)

RESULTS: (none)

MODIFIEDS: m (the original matrix with row r

removed, and all rows moved up by one)

INTERMEDIATES: index (index of row being moved)

HEADER: deleteRow(m, n, r) *(Alternative algorithm using the array of references)*

BODY:



public static void deleteRow(int [ ][ ]m,

int n, int r)

{

}

## CPU

**Program Memory Exercise 9-7 (continued) –** Delete a Row  **Working mem Global Mem**

GIVENS: m *(a square matrix)*

n *(size of m)*

RESULTS: (none)

MODIFIED: m *(last row put to 0)*

INTERMEDIATE: index *(index of the column)*

HEADER: putRowToZero(m, n)

private static void putRowToZero(int [ ][ ]m,

int n)

{

int index; *// INTERMEDIATE*

}

# 

CPU

# Section 10 Exercises

* **Exercise 10-1:** What is the problem with the following solutions:
  + Each value is stored in a separate variable:
  + Put all the values into an array:

**Program Memory Exercise 10-2 – First Version of Student Class Working memory Global Memory**

OR

**public class Student**

**{**

**}**

**Student**

(no methods yet!)

**Program Memory Exercise 10-3 – Object Usage In Java Working Memory Global Memory**

CPU

**public class Section10**

**{**

**public static void main(String [] args)**

**{**

**Student aStudent;**

**aStudent = new Student( );**

**aStudent.id = 1234567;**

**aStudent.midterm = 60.0;**

**aStudent.exam = 80.0**

**aStudent.forCredit = true;**

**Student meToo;**

**meToo = new Student( );**

**meToo.id = 1234567;**

**meToo.midterm = 60.0;**

**meToo.exam = 80.0**

**meToo.forCredit = true;**

**}**

**}**

CPU

**Program Memory Information Hiding in the Student Class Working Memory Global Memory**

# 

**public class Student**

**{**

**// Attributes**

**private int id;**

**private double midterm;**

**private double exam;**

**private boolean forCredit;**

**private double finalMark;**

**// Methods**

**public int getId()**

**{ /\* insert code here\*/ }**

**public void setId( int newId )**

**{ /\* insert code here\*/ }**

**public double getMidterm()**

**{ /\* insert code here\*/ }**

**public void setMidterm( double newMark )**

**{ /\* insert code here\*/ }**

**public double getExam()**

**{ /\* insert code here\*/ }**

**public void setExam( double newMark )**

**{ /\* insert code here\*/ }**

**public boolean getForCredit()**

**{ /\* insert code here\*/ }**

**public void setForCredit( boolean newValue )**

**{ /\* insert code here\*/ }**

**public double getFinalMark()**

**{ /\* insert code here\*/ }**

**private void recalculateFinalMark()**

**{ /\* insert code here\*/ }**

**} // end of class Student**

CPU

**Program Memory Methods in the Student Class Working Memory Global Memory**

**public class Student**

**{**

**// attributes and other methods would go here**

**public void setMidterm( double newValue )**

**{**

**this.midterm = newValue;**

**this.recalculateFinalMark( );**

**}**

**public void setExam( double newValue )**

**{**

**this.exam = newValue;**

**this.recalculateFinalMark();**

**}**

**private void recalculateFinalMark()**

**{**

**this.finalMark = 0.2 \* this.midterm +**

**0.8 \* this.exam;**

**}**

**}**

CPU

# Section 11 Exercises

**Program Memory Exercise 11-1 – Using Constructors Working Memory Global Memory**

# 

**public class Section11**

**{**

**public static void main(String [] args)**

**{**

**Student aStudent; // reference variable**

**Student meToo; // another reference variable**

**Student bStudent; // a third reference variable**

●

●

**aStudent = new Student(1234567,60.0,80.0,true);**

**meToo = new Student(7654321,true);**

**bStudent = aStudent;**

●

●

●

**}**

}

**class Student**

**{**

**// ... fields would be defined here ...**

**public Student(int theId, double theMidterm,**

**double theExam, boolean isForCredit)**

**{**

**this.id = theId;**

**this.midterm = theMidterm;**

**this.exam = theExam;**

**this.forCredit = isForCredit;**

**}**

**public Student(int theID, boolean isForCredit )**

**{**

**this.id = theID;**

**this.midterm = 0.0; // a “safe” value**

**this.exam = 0.0; // a “safe” value**

**this.forCredit = isForCredit;**

**}**

**// ... Other methods ...**

**}**

CPU

**Program Memory Exercise 11-2 – Calculation of the Final Mark Working Memory Global Memory**

# 

**public class Student**

**{**

**// Attributes**

**private int id;**

**private double midterm;**

**private double exam;**

**private boolean forCredit;**

**private double [] assignments;**

**// Methods**

**public double calcAssignAvg()**

**{**

**}**

**public double getFinalMark()**

**{**

**}**

**} // end of class Student**

CPU

**Program Memory Exercise 11-3 – Arrays of Objects Working Memory Global Memory**

# 

**public class Section11**

**{**

**public static void main(String [] args)**

**{**

**Course aCourse;**

**aCourse = new Course();**

**aCourse.addStudent(123456,true);**

**aCourse.addStudent(654321,false);**

**}**

**}**



CPU

**Program Memory Exercise 11-4 – Using Class Variable and Methods Global Memory**

**public class Section11**

**{**

**public static void main(String [] args)**

**{**

**int anum;**

**Student aStudent; // reference variable**

**Student meToo; // another reference variable**

**aStudent = new Student(1234567,60.0,79.0,true);**

**meToo = new Student(7654321,54.5, 83.4, true);**

**for(anum=0 ; anum<5 , num=i+1)**

**{**

**aStudent.setAssignment(anum, 80.0);**

**meToo.setAssignment(anum, 65.0);**

**}**

**System.out.println(“The mark for student “ +**

**aStudent.getId()+ “ is “+**

**aStudent.getFinalMark();**

**Student.SetMidWeight(0.30);**

**Student.SetAssignWeight(0.15);**

**System.out.println(“The mark for student “ +**

**meToo.getId()+ “ is “+**

**meToo.getFinalMark();**

**System.out.println(“The mark for student “ +**

**aStudent.getId()+ “ is “+**

**aStudent.getFinalMark();**

**}**

**}**

**Working Memory**

**Terminal Window**

CPU

**Exercise 11-5 – Designing a Fraction class**

* What information do we need to store in a Fraction?
* What operations do we need?
  + [Aside from creating fractions, the only mathematical operation we will implement is addition of two fractions]

**Fraction**

**Program Memory Exercise 11-6 – Simplify Fraction to Standard Form Global Memory**

**Working Memory**

**private void simplify( )**

**{**

**}**

CPU

**Program Memory Exercise 11-7 – Method for GCD Working Memory Global Memory**

**// a class method**

**private static int gcd (int a, int b)**

**{**

**}**

CPU

**Program Memory Exercise 11-8 – Fraction Constructors Working Memory Global Memory**

# 

**public class Fraction**

**{**

**private int numerator;**

**private int denominator;**

**public Fraction(int n, int d)**

**{**

**}**

**public Fraction(int a)**

**{**

**}**

**Exercise 11-9 Displaying Fractions**

**public void display( )**

**{**

**}**

CPU

**Program Memory Exercise 11-10 – Adding Fractions Working Memory Global Memory**

**public Fraction addTo(Fraction operand)**

**{**

**}**

**Exercise 11-11: Adding an Integer   
to a Fraction**

**public Fraction addTo(int integer)**

**{**

**}**

CPU

**Program Memory Exercise– Using Fraction Working Memory Global Memory**

**class FractionsDriverClass**

**{**

**public static void main (String[] args)**

**{**

**Fraction f1, f2, f3;**

**f1 = new Fraction(6, 9);**

**f2 = new Fraction(24, 46);**

**f3 = f1.addTo(f2);**

**f1.display();**

**f2.display();**

**f3.display();**

**System.out.println(f1);**

**System.out.println(f2);**

**}**

**}**

**Terminal Window**

CPU