ITI1120 – Introduction to Computing I
Exercise Workbook
Fall 2012

Table of Contents

1. 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.

2. 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).

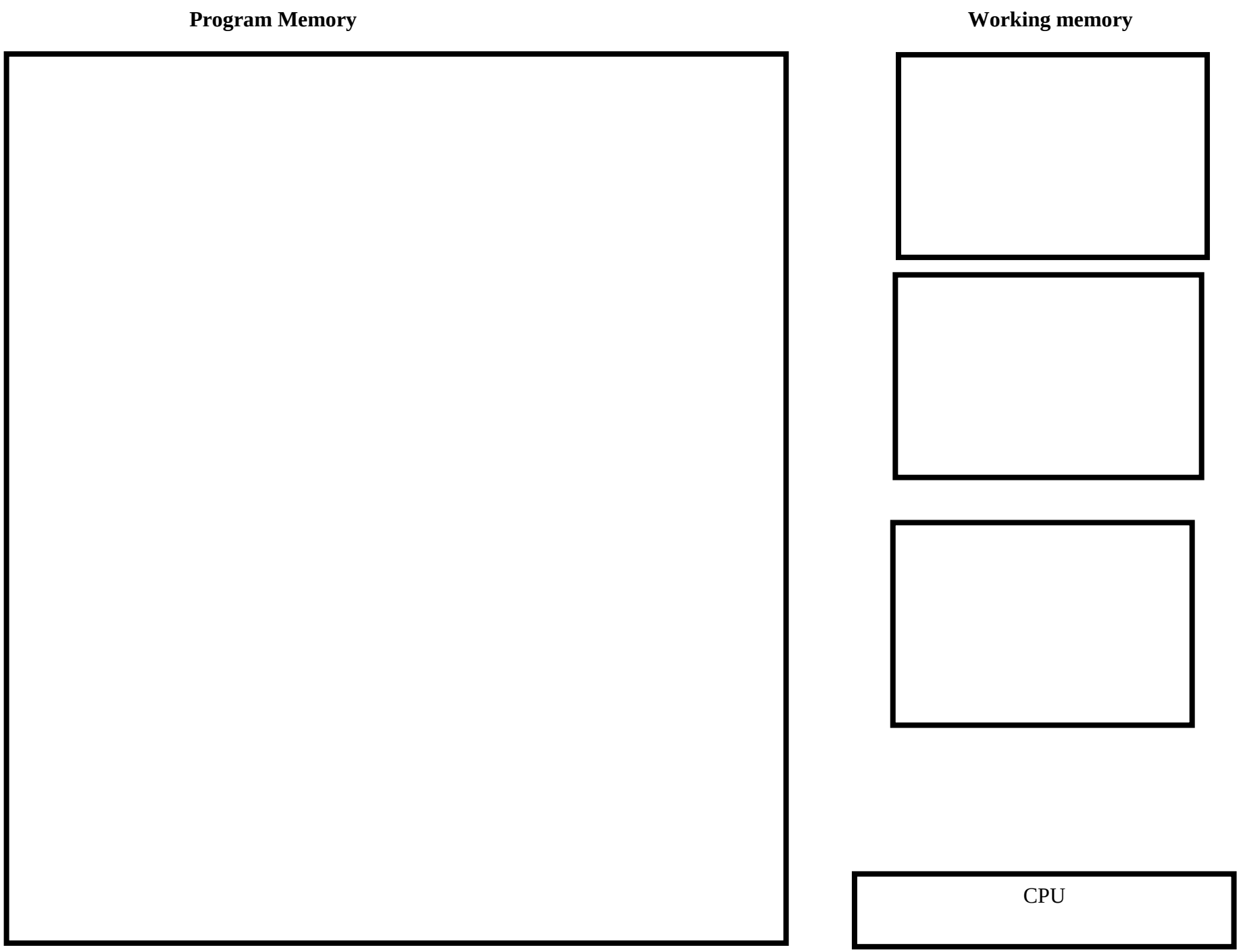
The machine instructions operate on variables, which are also located in the computers memory. Essentially the CPU reads values from and writes values to the variables (i.e the locations in memory where variables are located). Operations such as addition, incrementing, testing, are all typically done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

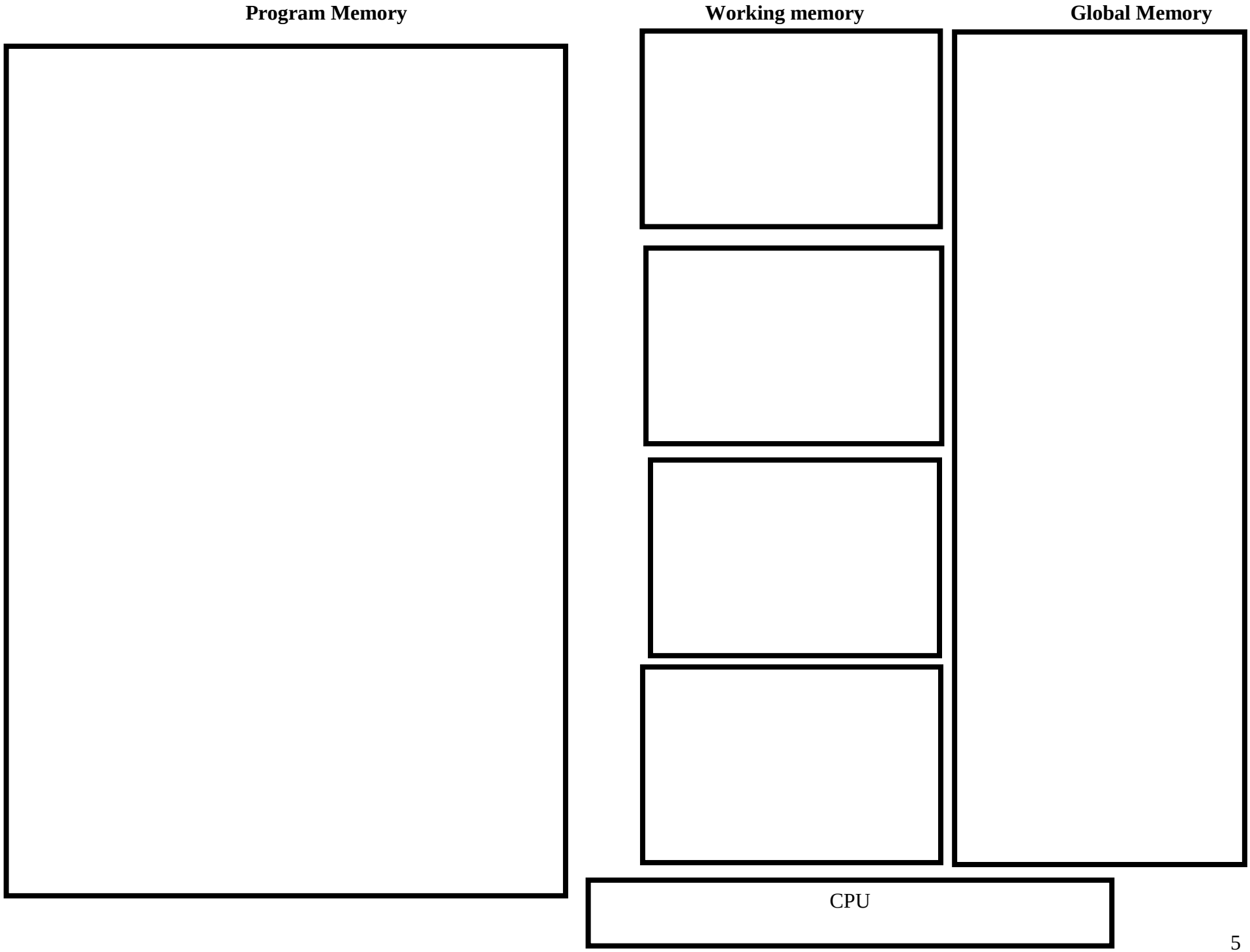
Computer memory consist of a sequence of bytes (8 bit values; a bit is a digit that can assume the value 0 or 1 and a byte contains 8 bits). Thus a computer that contains 3 GigaBytes of memory contains 3 x 1024 x 1024 x 1024 bytes, that is, over 3 billion bytes.

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

3. 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(L)
: Creates an array, referenced by anArrayRefVar, which has L positions with unknown values in them.

aMatrixRefVar ← makeNewMatrix(R, C)
: Creates a matrix, referenced by aMatrixRefVar, which has R rows and C 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.

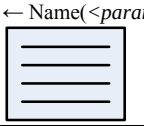
4. Summary of Program Structure, Algorithm Concepts and Java Programming Concepts

Algorithm Model

Instruction Block

Subprogram

…
Result ← Name(<*paramList*>)

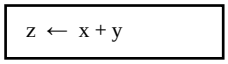


Instructions

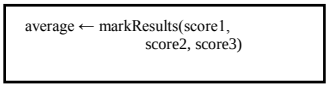
Operations

( ) *(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)*

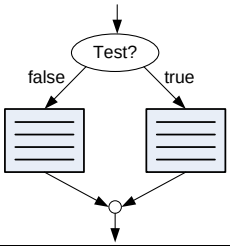
Simple Expression



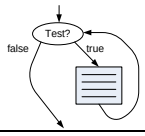
Calls to subprograms



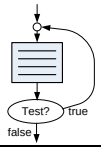
Branch Instruction



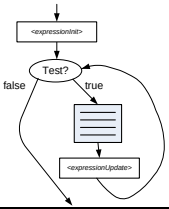
Pre-Test Loop Instruction



Post-Test Loop Instruction



Alternative Loop Instruction (Java **for** loop)



Java

Instruction Block

```
{
   <instruction>
   <instruction>
        .
        .
}
```

Subprogram

```
public static <type> name(<paramList>)
{
   <instruction>
   <instruction>
        .
        .
}
```

Instructions

Operations

( ) *(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

```
z = x + y;
```

Calls to subprograms

```
average = markResults(score1,
             score2, score3);
```

Branch Instruction

```
if (<logical expression>)
{
   <instruction>
   <instruction>
        .
        .
}
else
{
   <instruction>
   <instruction>
        .
        .
}
```

Pre-Test Loop Instruction

```
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>
        .
        .
}
```

Primitive Types

int	integers
double	real values
char	single characters
boolean	Boolean values

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
```

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();
```

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)
```