Object-Oriented Programming Fundamentals

Lecture/Workshop (Week 1)

Integer Division, Type Compatibilities

Task 1 - Fill in the values of the variables as the Java statements execute:

1 int voltage = 0, power = 0;
int current = 0, resistance = 0;

voltage	
power	
current	
resistance	

2 current = 5;

voltage	
power	
current	
resistance	

3 resistance = 2;

voltage	
power	
current	
resistance	

4 resistance = resistance * 3;

voltage	
power	
current	
resistance	

5 voltage = current * resistance;

voltage	
power	
current	
resistance	

6 power = current * voltage;

voltage	
power	
current	
resistance	

7 power = current * current *
resistance;

voltage	
power	
current	
resistance	

8 power = 1/2 * power;

voltage	
power	
current	
resistance	

Integer division

If a division involves two integers, the result will be an integer with the remainder discarded

Examples

```
8 / 4  ⇒  2
9 / 4  ⇒  2
```

Double division

If a division involves at least one **`double`**, the result will be a **`double`**

Examples

```
8 / 4.0  ⇒  2.0
9.0 / 4  ⇒  2.25
```

Conversion between double and int

Sometimes, we need to convert **`double`** values to **`int`** and vice versa

The conversion can be done with a type cast

Examples

```
9 / (double) 2   ⇒  4.5    2 is temporarily cast to a double
(double) 9 / 2   ⇒  4.5    9 is temporarily cast to a double
```

Task 2

What is the result of each of these statements?

```
int a = 9 / 2;

int b = 9 / 2 * 4;

double c = 9 / 2 * 4;

double d = 9 / 2.0 * 4;

double e = (double) 9 / 2 * 4;

double f = 9 / 2 * 1.5;

int g = 9 / 2 * 1.5;
```

Type compatibilities

In general, a variable of one type cannot store a value of another type

Examples

```
int counter;
```

```
counter = 2.34;
```
Incorrect (compile-time error)

```
counter = 'a';
```
Not incorrect but *poor style (counter is assigned the Unicode number representation for 'a' which is 97)*

Mixing numeric types

Java allows the mixing of **byte**, **short**, **int**, **long**, **float**, and **double** in arithmetic expressions

- If one argument of a binary operator is a **double**, the other argument is converted into a **double**, and the result is a **double**
- Otherwise, if one argument is a **float**, the other will be converted into a **float**, the result is a **float**
- Otherwise, if one argument is a **long**, the other is converted into a **long**, the result is a **long**
- Otherwise, if one argument is an **int**, the other is converted into an **int**, the result is an **int**
- Otherwise if one argument is a **short**, the other is converted into a **short**, but the result is an **int**
- In the case the two arguments are **bytes**, the result is still an **int**

So given:

```
double d = 1.5;
float f = 1.8F
byte b = 121;
int i = 2000000000;
```

Do the following compile, if so, what is the type of the result of the following operations:

i) d + f

ii) f + b

iii) b + b

iv) i * 2

Mixing with char

- Java allows the mixing of **`char`** with numeric data types
- A **`char`** argument of a binary operator is always treated as an **`int`**. Thus the result is an **`int`**

So given:

```
double d = 1.5;
byte b = 121;
char ch = 'A';
```

do the following compile, if so, what is the type of the result of the following operations:

i) **`d + ch`**

ii) **`b + ch`**

iii) **`ch + ch`**

Mixed types in assignments

As a special case, we can assign an **`int`** value to a **`double`** variable; but not vice versa

In general, Java performs the following implicit type conversions for assigning a value to a variable of a different type

`byte` → **`short`** → **`int`** → **`long`** → **`float`** → **`double`**
`char` → **`int`** → **`long`** → **`float`** → **`double`**

These are all considered widening conversions as they convert the data into another type that represents a wider range of numbers; the magnitude range of the data will not be lost

In the case of converting an integer type to a floating point type, some precision may be lost

Given

```
double x = 1.3;
int i = 3;
byte b = 5;
char ch = 'B';
```

are the following statements allowed and if so, what is the value assigned to the lhs (left-hand side) variable

i) `d = i;`

ii) `i = d;`

iii) `d = ch;`

iv) `b = ch;`

v) `ch = b;`

Explicit type casts

When it is required by the programming logic, we can explicitly convert a data value of one type to another type

When converting a data value stored in one type to a type that represents a narrower range of numbers, information can be lost or unexpected results may occur

These conversions are referred to as narrowing conversions

To make a narrowing conversion we have to explicitly tell the compiler with a type cast

What is the value in i?

```
double x = 12.34;
int i = (int)x;
```

What is the value in b?

```
int i = 256;
byte b = (byte)i;
```

Type name	Memory used	Possible values	Size range
byte	1 byte	2^8	-128 to 127
short	2 bytes	2^{16}	-32768 to 32767
int	4 bytes	2^{32}	-2147483648 to 2147483647
long	8 bytes	2^{64}	-9223372036854775808 to 9223372036854775807

Type name	Memory used	Size range	Precision
float	4 byte	-3.4×10^{38} to 3.4×10^{38}	≈ 7 sig. digits
double	8 bytes	-1.7×10^{308} to 1.7×10^{308}	≈ 15 sig. digits

Character	Code	ASCII Value
A	\u0041	65
B	\u0042	66
C	\u0043	67
D	\u0044	68
E	\u0045	69
F	\u0046	70
G	\u0047	71
H	\u0048	72
I	\u0049	73
J	\u004A	74
K	\u004B	75
L	\u004C	76
M	\u004D	77

Character	Code	ASCII Value
N	\u004E	78
O	\u004F	79
P	\u0050	80
Q	\u0051	81
R	\u0052	82
S	\u0053	83
T	\u0054	84
U	\u0055	85
V	\u0056	86
W	\u0057	87
X	\u0058	88
Y	\u0059	89
Z	\u005A	90

Task 3

Consider the following three programs. Determine which programs will work.

- If it will not work, what can be done to allow the program to work?
- If it will work, what would be output by the program?

```
public class WhatsGoingOn
{
   public static void main(String[] args)
   {
      int intNumber = 0;
      double doubleNumber = 7.59;
      intNumber = doubleNumber;
      System.out.println("intNumber: " + intNumber);
   }
}
```

```
public class ChangingChar
{
   public static void main(String[] args)
   {
      int number = 76;
      char character = 'E';
      System.out.println("character before assignment: "
                         + character);
      character = (char) number;
      System.out.println("character after assignment: "
                         + character);
   }
}
```

```
public class Division
{
   public static void main(String[] args)
   {
      int num1 = 0, num2 = 0, num3 = 0, num4 = 0;
      double num5 = 0, num6 = 0, num7 = 0;
      num1 = 4 / 2;
      num2 = 5 % 4;

      num3 = num1 + num2;
      num4 = num3 * num3;
      num5 = num4 / 2.0;
      num6 = num4 / 2;
      num7 = (double) num4 / 2;
      System.out.println(num1 + " " + num2 + " " + num3);
      System.out.println(num4 + " " + num5 + " " + num6);
      System.out.println(num7);
   }
}
```

Steps in Program Development[1] – background reading

(to be completed by you)

What is computer programming?

… the process of developing a user-friendly product that solves a user's data processing problem by developing and writing an efficient computer program

To program a computer, we need to see the world in terms of *data* and *processing*.

To write a program, programmers need to:
- Talk with clients to find out their requirements
- Understand the problem to be solved
- Design a solution to the problem
- Implement (program) the solution
- Test the solution to the problem
- Enhance and maintain the solution

So, a lot of activity occurs *before* fingers touch the keyboard…

In our subject, we will focus on the following steps involved in solving problems on a computer

1. **Understand the problem**
2. **Design a solution**
3. **Implement (program) the solution**
4. **Test the solution**

Step 1. Understand the problem – Defining Diagrams

A *Defining Diagram* is a technique to help *understand the problem*. It provides an overview of a problem in table form, showing:
- what data needs to be supplied to solve the problem (*Inputs*)
- what needs to be done (*Processing*)
- the results (*Outputs*)

Inputs	Processing	Outputs

Example – consider the following problem

Input two numbers, add the two numbers together and display their total.

Example Program Run

```
Run
Number 1 ? 10
Number 2 ? 3
Total = 13
```

[1] Adapted from CSE1PE (Programming Environment) materials – ref: L.A. Robertson, *Simple Program Design*

Steps in developing Defining Diagrams

Step 1: Study the problem

Think about inputs, processing and outputs. *Inputs* and *outputs* are usually the things being acted on and are usually referred to by descriptive words, such as *two numbers* and *total.*

> Input two numbers, add the two numbers together and display their total.

The *processing* required is usually indicated by verbs - indicating the actions to be performed, such as *input*, *add* and *display*.

> Input two numbers, add the two numbers together and display their total.

Step 2a: Determine the outputs (result)

It is often easiest to identify the output(s) for a problem. In this case the user wants to know what the total of the two numbers is.

Step 2b: What processing is required to obtain the result

The two numbers must be input, added together and the total displayed.

Step 2c: What inputs are needed

What is the least amount of information that must be supplied by the user. If the information can be provided by another source (e.g. calculation) then don't ask the user. The two numbers to be added can only be obtained from the user, there is no other way to get them.
Let the inputs be number1 and number2 (or other suitable names).

Step 3: Develop the Defining Diagram

Inputs	Processing	Outputs
number1, number2	Input number1, number2 Add the numbers together Display the total	total

---

Task 1

Fill in the defining diagram below for the following problem.

> Input the length and width of a rectangle (in cm), calculate the perimeter and display the perimeter.

Inputs	Processing	Outputs

Step 2. Design a solution – Algorithms (Pseudo Code and Data Dictionaries)

A Designing Diagram expresses a problem at a high level. Details not in the high level solution can now be given in a solution outline that should be developed into an *algorithm*.

An *algorithm* is a series of steps to be performed to solve the problem. Common examples include recipes, instructions for assembling furniture and directions to a destination. We will use *Pseudo Code* and *Data Dictionaries* to help write our algorithms.

Pseudo Code

There are no strict rules for how to write the pseudo code to solve a problem. Pseudo Code (which means *fake* code) consists of short English phrases to be read by humans (not by a computer). Although there are no strict rules for writing Pseudo Code, we will follow certain conventions in this subject to show the steps required and the order they should be done in.

A simple example where the steps to be performed are done in sequence (that is the flow of control is to do the steps one after the other) follows for the problem:

Input two numbers, add the two numbers together and display their total.

```
AddTwoNumbers                                  name of the program
    Input number1, number2
    total = number1 + number2                  steps to be performed / actions
    Display total
STOP                                           after the actions, the program stops
```

Data Dictionaries

Data Dictionaries describe the data used in the Pseudo Code (that is, the data used to solve the problem). A Data Dictionary is given in table form, showing:
- the identifier, such as a variable name (*Name*)
- the type of the data, such as Integer (*Data Type*}
- a brief, meaningful description of the identifier (*Description*)

Name	Data Type	Description

In the description, specify units of measure if appropriate, and examples can be given for clarity. The rows are sorted in alphabetical order on the identifier name (the first column). The Data Dictionary for the example above is:

Name	Data Type	Description
number1	Integer	The first number to be added
number2	Integer	The second number to be added
total	Integer	number1 and number2 added together

Task 2
Write the Pseudo Code and Data Dictionary for the problem from Task1.

Pseudo Code

Data Dictionary

Name	Data Type	Description

More on Step 2. Design a solution – <u>Desk Check</u>
Note that an algorithm should be checked to see if it meets the users' requirements. As the program is yet to be coded, the algorithm is manually checked (on paper).

Consider our algorithm (Pseudo Code with line numbers for reference).

```
1. AddTwoNumbers
2.       Input number1, number2
3.       total = number1 + number2
4.       Display total
5. STOP
```

We can draw a table with columns for:

- Pseudo Code *Line Number* (to specify the line(s) being executed)
- *One column per variable* used. The columns should be in alphabetical order on variable name. As the algorithm is executed, the new values of the variables are put in the appropriate column. Show working for calculations.
- *Input/Output* to show what is input by the user and displayed by the program. Show inputs with the variable name, a "?" and the value input e.g. price ? 200. Show outputs with the variable name, an =, and the value displayed e.g. discountPrice = 180

Desk Check

Inputs: number1 = 10, number2 = 3
Correct result: total = 13

Line Number	number1	number2	total	Input/Output
1,2	10	3		number1?10 number2? 3
3			10+3=13	
4				total = 13
5				

Task 3
Desk check your algorithm for the problem from Task 1.

Desk Check

Resources
Under *Guides / References* on the LMS (Moodle) area for CSE1OOF/4OOF, the following guides can be found:

- Defining Diagram Guide
- Pseudo Code Conventions OOF
- Data Dictionary Guide
- Desk Check Guide

This worksheet only looks at a very small part of these guides. However, you will want to refer to these during the semester, so please check out where they are, for later use.

Unix Operating System[2]

- Multitasking, multi-user Operating System
- A user is a person with an account on a machine
- A userid or username is a unique name for a user's account on a machine
- Each account has a password which is a secret code required to access it
- An account has details associated with it such as an expiration date and an amount of disk space that it is allowed to use

Files and directories

A computer stores all its information in files. These files are kept in a structure called a directory. A directory can hold any number of files and also other directories. Directories are used to group files together which have a common relationship.

Directories and files are arranged in a hierarchical tree structure with the top of the tree known as the root. The '/' symbol refers to the root directory in Unix systems.

Each account has a home directory where creating and deleting files and directories is allowed.

On initially logging into an account, the current working directory is set to the account's home directory.

Files and directories are referenced using their name. In Unix, both file and directory names are strings of characters with little restrictions on the available characters. You cannot use the '/' symbol as it already refers to the root directory. However it is also recommended that you do not use spaces, tabs or any other non-printable characters in your file or directory names as later manipulation becomes difficult.

Unix (Linux) Programming Environment

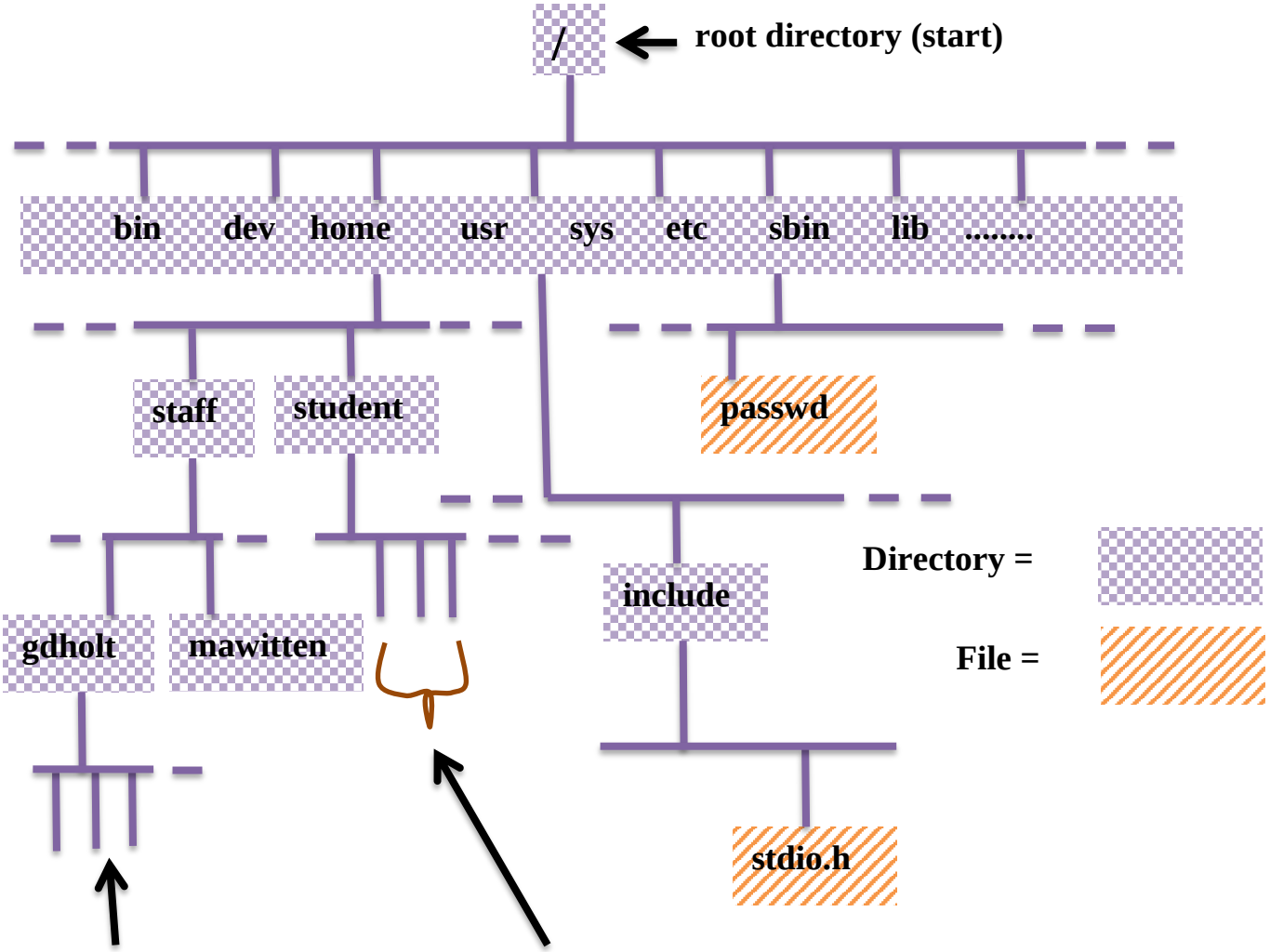
You will have an account on the department's Linux machine, **latcs8**. Some of the directories and files on latcs8 are shown on the next page[3]. You will be introduced to this programming environment in your first lab.

You will be able to create directories and files in your home directory on latcs8.

It will be your task to group your files in a sensible manner!

[2] Reference: H. Hanh, *Harley Hahn's Guide to Unix and Linux*
[3] Adapted from CSE1PE (Programming Environment) materials

Partial Unix File System (from latcs8)



directories and files under the control of gdholt

The home directories of the students doing our subjects - one is your (home) directory with your login id. You will have control of the directories and files in your home directory.

Task 4

Peter Piper is a student who's just started 3rd year. Over the past two years he hasn't bothered to use directory structures in his university account and so it contains many files all in the one directory. He finds that he often tries to create files but has already used that name, and he often cannot find existing files that he wants for a particular purpose.

Peter has files from his first two years at university during which time he took 3 programming subjects (CSE1OOF, CSE1IOO and CSE2ALG) and 2 maths subjects (MAT1DM and MAT2MCS). For the programming subjects he has code from each of the 12 lab classes and 1 assignment. He also has files which contain code given in the lectures which he typed in and tested. He has several word documents for the 4 assignments in each of the two maths subjects. He also has some personal photos, a couple of job applications and several letters he wrote home to family.

Draw a directory structure that Peter could use to organise his files (starting from his home directory called **ppiper**).

What criteria did you consider when creating the directory structure?

Absolute vs relative pathnames

The **absolute pathname** of a file is the full name of the file, showing every directory from the root (top-level) to the file. In Linux the absolute pathname will always start with the '/' character which indicates the root directory.

A **relative pathname** of a file shows the path to the file starting from the current directory.