Week 1

Sequence, Variables, Input and Output
Class exercises

1 Ceebot Task 1.6: Power up a Robot (use variables)

Notice that you can zoom in and out of a scene, by using the **+** and **-**

buttons

If you zoom onto the **WingedGrabber** robot you will see that its energy level is zero (shown by the green indicator). This is because it has no power cell on board.

Your task is to get a powercell to this robot using the other robot .. a **WheeledGrabber** .. that can pick up and carry things.

- So now you must program the WheeledGrabber robot, using the **editor** to create the program, as in the previous exercises.
- The following instructions will be useful for this task: (use **[F2]** for more details)
  - **move()** .. move forward .. the parameter is the distance **(use a variable)**
  - **grab()** .. picks up any object in front .. it has no parameter
  - **drop()** .. drops the object being carried .. normally has no parameter
  - **turn()** .. the parameter is the angle passed to it .. **(use a variable)**
- Work out the program algorithm by yourself. **[F1]** will show a picture of the scene with some useful distance information.
- When you have been successful, the WingedGrabber will fly off, because it already has been pre-programmed to do a task of its own.

Note: when the WingedGrabber flies off, you can follow it if you want to .. just click on its icon at the top left of the screen .. then try zooming and using the camera button too.

2 Ceebot Task 4.1: Follow a Path (using variables)

This time you are on a more peaceful planet. There is a different robot here, a **WheeledGrabber**.

Your task is to program the robot to reach the platform that you can see in the distance. The path to take is marked out with blue waypoints (checkpoints). You may like to know that these are all **20 metres** apart.

- So now you must program the robot, using the **editor** to create the program, just like you did in the previous exercise.
- You will need to use the **move(…)** and **turn(…)** instructions.
- Notice that most programming instructions have brackets. Inside the brackets you can put a value (called a **parameter**) and this value is then passed to the instruction. In this case the parameter is the length the robot moves or the angle it turns.
- It is generally good programming practice to use **variables**, so start by setting up some variables for the length and angle:

```
float  length;
float  angle;
```

- Then work out the **algorithm** steps in normal English (see the start made here:)

Algorithm
1. **Set length to ??**
2. **Set angle to ??**
3. **Move forward using length**
4. **- - - - - - -**
5. **- - - - - - -**

etc.

- Then code your program and get it to work (put algorithm and code in the logbook)

3 Ceebot Task 6.3: Step by Step

This is another example of moving, but this time you have to use the dialog box to input numbers and also convert them from a string type to a number type. Your WheeledGrabber is at one end of the track and there are mines at the other end, just past a platform .. your destination.

Your task is to reach the platform and survive, by asking the user to **input** a distance to move, then to move the robot through that distance. If you keep running the program, you should eventually reach your destination.

- You will need the **dialog(..)** instruction and also the **strval(..)** instruction to convert your string input into a number .. see the start of this unit for details.
- You should also try to input a distance to travel to the platform in **one** go!
- Unfortunately each time you reset the program, the platform distance will be different!
- Your main algorithm is:

Algorithm
1. **input a distance from the keyboard**
2. **convert this to a number**
3. **move this distance**

Put the code into your log book

4 Task 5.3: RoboMaths

If you are not so good at maths, have no fear because these robots, like all computers, are very good (and fast) at doing calculations .. just so long as you know how to program them correctly!

Your Task

To get you into the swing of things you are to program this robot to do some fairly straightforward arithmetic. To do this we always use **variables** because then the same program can be used with different numbers.

- **Use the algorithm here for your program.**
- The final display should look like this:
  The answer is …
- you will need to use the **message(…)** instruction for the final display. (see your lecture notes or hit the [F2] key)

Algorithm

1. **Store 2 in variable a**
2. **Store 8 in variable b**
3. **Subtract 2 from variable b**
4. **Multiply a by b and store result**
5. **Display the result**

Extra

Program the final message display so that it shows the contents of **all** the variables, like this:

Multiplying ... by ... gives …

5 Ceebot Task 6.4 (variation) : Think of a number!

Use the algorithm below to write the program:

Algorithm

1. Input any value from the user and convert this to a float number
2. Double this number and store the result separately
3. Add 16 to the previous result
4. Divide the result by 2
5. Subtract the original number from this result
6. Display the final answer

Now run the program again, entering a different starter value .. what do you notice?
Can you explain why the program works this way?

Put source code and comments in the logbook

6 Ceebot Task 6.5: Target Practice

You are now going to set up a program that can be used to do some target practice. When your shooter has moved into position, you will have a 'robot's eye' view of the scene with 5 explosive targets in front of you.
The shooter's cannon has two settings that are important:

- the vertical angle set by the **aim(…)** instruction (this can be between –20 and 20)
- the horizontal angle used by the **turn(…)** instruction (between –90 and 90 here)

Your task

- write a program to input the horizontal and vertical angles and then use these to destroy one of the targets.
- Your cannon should then be returned back to its starting position, ready for the next run of the program

Here is an **algorithm** that should help:

Algorithm

1. **Input the vertical angle (-20 to 20)**
2. **convert this to a float number**
3. **Input the horizontal angle (-90 to 90)**
4. **Convert this to a float number**
5. **Aim the shooter**
6. **turn the shooter**
7. **Fire**
8. **Set the aim back to horizontal**
9. **Turn back to the starting position**
10. **Output a message "Ready to fire again"**

Testing the Program

Again a **Test Plan** has been partially completed for you.

- Run the program 3 times using the input data for the 3 existing tests and fill in the results.
- Then work out the values for Tests 4 and 5 and run these to complete the testing

Shooting Practice
Test Plan

Test No.	Input Angles		Expected Result	Actual Result
	Vertical	Horizontal		
1	1	7	Target destroyed	
2	8	30	Target destroyed	
3	-3	-16	Target destroyed	
4			Target destroyed	
5			Target destroyed	

Finally put your source code and the completed Test Plan into your log book

Week 1: The Technical Bit

1. Sequences

The **sequence** is a very important programming construct.
Basically, a sequence is a group of instructions or statements, where one instruction follows another in a particular order..
These instructions all end with **semicolons** and are put inside a set of **braces** to make what is known as a **block** of instructions .. like this:

```
{
        -----  instruction 1 ;
        -----  instruction 2 ;
        -----  instruction 3 ;
        -----  instruction 4 ;
}
```

The **order** of the instructions is very important and the skill of the programmer lies in putting the correct instructions together in the right order to achieve the task. All the programs you have written this week are sequences.

2. Algorithms (Pseudocode)

As programs get larger, it is very easy to get the order of the instructions wrong.
Good programmers design their programs before getting down to detailed coding. One of the techniques used to do this is the **algorithm** (also known as **pseudocode**)
An algorithm is just a plan for the program, written in structured english, usually with the various steps numbered in the correct order.
For example:

1. move 20 metres forward
2. turn 45 degrees right
3. pick up object
4. turn 180 degrees
5. move 10 metres forward
6. drop obect
7. pause for 2 seconds

3. Variables

Often in a program, you want to store some value that may change later on in the program. Using a variable allows you to do this.

A variable is like a storage box .. you can have many boxes of different sizes … and so you don't forget what is in the box, you should give it a sensible name (or identifier)

A **variable** is a temporary storage 'box' in a program. You can use variables to store numbers, text, etc. (see next page)

Declaring Variables

Every variable in a program needs to be set up by defining or **declaring** it before it can be used. This means giving the variable a **type** and a **name** (also called its **identifier**) : e.g.

```
int  number ;          // defines number as a variable able to store an integer
float  wage ;          // defines wage as a variable able to store a decimal
                          (or floating point) number
string  surname ;   // defines surname as a variable able to store a string of characters
object  item ;         // defines item as a variable able to store object details
boolean  found ;    // defines found as a variable that can only be true or false
```

Assigning Values to Variables

Variables can be assigned values using the **=** operator

e.g.

```
number = 204;        // stores 204 in the number variable
wage = 20.25;        // stores 20.25 in the wage variable
surname = “Tita”;      // stores Tita in the surname variable
wage = wage * 2;     // changes the number stored in wage to 40.50
wage = number + wage + 10;     // stores 254.50 in the wage variable
(204+40.50+10)
```

All of the above are examples of assignment statements

Output

Often you want to output information to the screen. In Ceebot this is done using the **message(…)** instruction .. this will print a message box onto the screen .. it will disappear in a few seconds. Here are some examples:

```
message("Hello Everyone");
message("My name is " + firstname + " and my age is " + age);
```

Notice you use "quotes" for strings of characters, but not for variables
Notice how the **+** operator is used to join strings and variables together into one message.

Input

You can also input information from the keyboard during a program. To do this you use the **dialog(…)** instruction. You will need to set up a string variable to store the input.
e.g.

```
string  input;                              // set up a string variable called input
input = dialog("Where do you live?");    //  input words from the keyboard
```

Notice the dialog(…) instruction has a string parameter .. this is a **prompt** to the user and appears in a dialog box on the screen.

Note that if you want to input a **number** to be used later in the program, you will have to convert the input string to a number .. you can use **strval(…)** to do this. e.g.

```
string  input;        // set up a string variable called input
int  number;          // set up an integer variable called number
input = dialog("How many Targets can you see?");   //  input from keyboard
number = strval(input);    // convert input to a number
```

Week 1: Independent Study (4 Tasks)

The following exercises are for extra practice .. to be completed during the week .. outside the normal classroom session. Put code and algorithms in logbook.

8 Ceebot Task 3.4: Production Chain (use variables)

In this scene, you can see a **WingedShooter** with no power cell. There are NO working power cells nearby and no Titanium cubes either!

- there is a **Converter** and this is able to convert Titanium Ore into Titanium Cubes.
- then the **Power Cell Factory** can finish the job by producing a power cell from the Titanium cube.

Your task

- Program the **WheeledGrabber** to pick up the Titanium Ore and deposit it onto the Converter. The Converter will take about 15 seconds to make a Titanium cube .. and you will have to move out of the way to let it do its job.
- Then take the cube to the PowerCell Factory … after the Power Cell has been created (about 5 seconds) you will need to install it in the WingedShooter like you did before.
- You will need to plan these steps carefully in order to succeed.
- Note that all the objects are **3 metres** from your current robot position
- You should use **variables** for your solution, where possible.

You should do this task in 2 stages:

- First work out the **algorithm** steps in normal english
- Code the program and get it to work

You should put both algorithm and code in your log book

9 Ceebot Task 4.3: Using a Radar

Your task is to bring TitaniumOre to the Converter so it can be converted into a Titanium cube. This sounds simple, but where is the Ore? Every time you reset this task, the Ore is in a different place! To find it, use the robot's **radar**.

- The **radar** represents the "eyes" of a robot. It is used to detect **objects** around it, even when they are hard to see. The parameter of the radar instruction is the object category to be detected and the radar's results are stored in an object variable.
- We can declare an object variable like this:

  object item; // item is now an object variable
- Then we can use the variable to detect an object, like this:

  item = radar (TitaniumOre); // detect the closest TitaniumOre object

  item now holds all the information about the object

- Use the **goto(…)** instruction to go to the item position like this: **goto(item.position);**

Use all this new information to complete the task, using this algorithm to help you:

Algorithm
1. Use radar to find position of a TitaniumOre object
2. Go to this position
3. Pick up the object
4. Use radar to find position of the Converter
5. Go to this position
6. Drop Titanium onto Converter
7. Step back to allow converter to do its job

10 Ceebot Task 5.5: Draw Rectangles

Your task is to write a program where the user is **asked** what length and width they require for to draw a rectangle. The program should then try to draw the required rectangle.
So a typical execution of the program could look like this:

What length (in metres) would you like for the rectangle?
2
What width (in metres) would you like for the rectangle?
3

At the end of the program a message should be displayed saying:

Rectangle of length 2 **metres and width** 3 **metres completed.**

Testing
Create a proper **test plan** for your program with at least 3 tests .. then test your program using your plan

You should do this task in 3 stages:

- First work out the **algorithm** steps in normal english
- Code the program and get it to work
- Test the program using your test plan

You should put **algorithm**, **code** and completed **test plan** for the program into your log book

11 Ceebot Task 5.2: Target Buster 2

There are two targets out there, but you don't know where they are. Again you can use your radar to find them.

- The category to use as parameter for the radar is **TargetBot**.

Your task is to find and destroy the two targets.

- However, you mustn't get too close to the explosive target or your robot will be destroyed and you won't get a chance to fire.

See Next page for a possible algorithm

- One possible **algorithm** to use would be this (to be done twice):

Algorithm (copy twice)

1. **Use radar to find TargetBot details**
2. **Store target's position**
3. **Store robot's position**
4. **Calculate distance from robot to target**
5. **turn towards target**
6. **Move robot to a safe firing distance (within 15 metres)**
7. **Fire for 1 second**
8. **Pause to ensure target destroyed**

New Information and instructions you need

- Position details for objects and robots can be stored in **point** variables .. for example:

```
point  botpos;                  // declare a point variable called botpos
point  targetpos;               // declare a point variable called targetpos

botpos = this.position;         // store robot (this) position
targetpos = item.position;      // store target position (after radar used)
```

The **distance(… , …)** instruction

- Distance can be calculated using the **distance(…)** instruction. It has 2 parameters and calculates the distance between 2 points. For example:

```
dist = distance (botpos, targetpos);   // calculate distance between 2 points
```

(n.b. dist must first be declared as a float variable)

The **direction(…)** instruction

- Direction or angle can be calculated using the **direction(…)** instruction. This has one position parameter and works out its angle from the robot.
  You can then **turn** using this direction: For example:

```
turn( direction(targetpos) );      //  turn in the direction of the target position
```

Now put all this together to finish the task and put the code in your logbook