Week 3

Selection

The Technical Bit

Selection is the third important programming construct, along with **Sequence** and **Iteration**

1. Selection using if(…)

There is often a need to make choices in a program, so that the program will perform some actions(once) if a certain condition is true.
This is programmed using the **if(…)** instruction. Example:

```
if  (count < 10)        // if count less than 10
{
     // ---- put instructions here to be done once if the condition is true
 }
```

In this example, when the program reaches this section of code, it will perform the block of instructions inside the braces **{ }** … but **only** if the **condition** (count less than 10) is **true**. Otherwise it will just carry on with the rest of the program.

2. Selection using if (…) else …

```
if (count < 10)       // if count less than 10
{
    // ---- put instructions here to be done once if the condition is true
}
else
{
    // ---- put alternative instructions here to be done once if the condition is false
 }
```

This time the block of instructions inside the first braces **{ }** is done only if the **condition** (count less than 10) is **true**. If the condition is **false**, the **else** set of instructions is done.

1

Ceebot Task 9.1: Spare Me!

Here we have a shooting exercise with 6 targets to destroy. The targets are 5 metres apart … but .. oops .. the astronaut is in one of the target positions!.

Your task:
Use a **for loop** to destroy the targets, but avoid the third position by using an **if(…)** instruction.

- Normally, the robot will move forward, turn, fire, and then turn back.
- This would be repeated 6 times.
- But now we should only do this if we are not at the third position.

A suitable **algorithm** for the program is shown here:

Algorithm
1. **Loop 6 times**
   a. move forwards 5 metres.
   b. **if** we are NOT at position 3
      i. turn left 90 degrees.
      ii. shoot.
      iii. turn right 90 degrees.
   End if
End Loop

Extra
Devise another method of tackling the same exercise. Press **[F1]** for a second algorithm.

2

Ceebot Task 9.2: Spare Two

Oh dear, this time there are 2 positions to be avoided .. positions 2 and 5 have engineers working away. Don't shoot them!

Your task:
Use a **for loop** to destroy the targets, but avoid the second and fifth positions, using ONE **if(…)** instruction.

- Because there are now **two** conditions to use in the if statement we can combine them using the **&&** operator (**AND**) .. see below.
- You could also use the **||** operator (**OR**)
- Design a new algorithm for your program and get it to work

Examples using && and || operators

```
if (count > 0 && count <=10)
{
    // do something if both conditions are true (count > 0 and count <=10)
}

if (count == 1 || count == 8)
{
    // do something if either condition is true (count == 1 OR count == 8)
}
```

3 Ceebot Task 9.4: Power Up?

There are 6 WingedShooters ready to fly .. some of them have no power cells while others need their power cells replacing. So there are two different situations to deal with .. a suitable case for an **if … else …** statement.

Your task:
Use a **for loop** with an embedded **if .. else ..** statement .

A partial **algorithm** is shown here:

Your first step should be to complete this algorithm:

- Note which robots have no cells at all and which need their cell replacing.
- Then work out what the **condition** should be for the **if** statement (you may need to use **||** operators)
- work out the steps needed to supply a new cell
- work out the steps to replace a cell

Algorithm (part completed)
1. **Loop 6 times**
   a. **if** (*condition*)
      ... supply a new cell
   else
      ... replace existing cell
   end if
End Loop

Note: the power cells on the ground are all 3 metres apart

4 Ceebot Task 7.6: Calculator 2 (Extension)

Your task: Extend your code from last week so that the program validates what the user inputs, only allowing numbers. If the data the user enters isn't a number, the program should output a message saying that this input isn't a number.

The idea here is to stop the program from crashing (or an error) if a wrong type of data is entered, and also offer the user a chance to re-enter a number.

Hint: write a do while loop, with an if statement inside that tests whether the input is a number. Also pay attention to what **strval()** returns when trying to convert non-numerical data to a numerical format…

Test the program by trying to input a letter, or a word, or a symbol, and see whether your validation code prevents the program from crashing.

5 Ceebot Task 10.6: Invisible Enemy Attack 2

You have just arrived on a Spaceship and your **WheeledShooter** robot is keen to get home to your village in the distance. This looks like a peaceful scene but your robot senses danger and stops!

Your task 1:

- Program your robot to move the 80 metre distance to the village.
- Run the program and the danger should then become all too clear.

Your task 2:

- Now your task is clear .. to reach home after getting rid of the sneaky enemy invaders along the path. Fortunately they always appear in the same positions (see below)
- You will need to **turn()** and **fire()** in a similar way to previous exercises but note that the **move()** instruction has to finish before you can do anything else .. so what do you do?
- One solution is to move your robot in smaller chunks .. for example **5 metres** at a time .. if you do this **16 times** you will reach the home village (80 metres away) and you can turn and fire if necessary (note: firing for 3 seconds is advisable to destroy some robots!)
- So program a **while loop** that repeats 16 times
- Inside the loop you can:
    - move 5 metres
    - use **if statements** to check your position and fire if necessary (see below)
    - Note that some robots are to the left and some to the right!

Enemy Positions

- The enemy robots appear at 15, 30, 35, 45, 55 and 65 m. along the path
- These are positions 3, 6, 7, 9, 11 and 13 .. if moving 5 m. a time and using a loop counter

6 Ceebot Task 10.3: To Be or Not to Be?

This scene looks familiar , with 10 possible target positions, but each time you reset the program the target positions will be different. We are going to tackle the program differently by asking the user whether to fire or not.

Your task:

You are to destroy all the Targets while avoiding the engineers and their Titanium cubes.

- The target positions are each 5 metres apart, so you must use a **for loop** to move your robot forward 5 metres at a time.
- After each move, you should use the **dialog(…)** instruction to ask :
  "Destroy (y/n)?"
- **If** the input answer is **"y"** the necessary instructions are performed to destroy the target, otherwise it is left alone.

Note: targets should be destroyed for any of the following : **"y" , "Y"**, **"YES"** or **"yes"**

Extra

1. You must output a **message** to the screen after each firing saying how many Targets have been destroyed so far.
2. You must also output **messages** showing the result of each action:
   e.g. either **Target** 3 **Destroyed** or **Target** 3 **Avoided**

7

Ceebot Task 9.7: Roll Call 1

You have 10 **WingedShooter** troops lined up in front of your robot, 5 metres apart .. but some of them are sick. As your robot moves forward to take the roll-call, they will either move onto their platform and report for duty or fly off to sick bay!
Note that the situation is different each time you reset the exercise!

Your task:
You are to count up how many of your troops are sick and how many are fit for duty.

- Start by programming a **loop** where your robot just moves forward 5 metres at a time along the row of troops.
- You will see the troops coming forward or flying off, but how can you count them?

Program Guidance

- You can use your **radar()** instruction to help you:
  item = radar (WingedShooter, -90, 10);
  it will send a narrow (10 degree) beam to the right of your robot (-90 degrees)

- If you modify this :
  item = radar (WingedShooter, -90, 10, 0, 6);
  the beam will only detect between 0 and 6 metres away. This will detect troops that moved forward to the platform, ignoring the ones that flew off!

- If the radar doesn't detect anything, it returns a **null** value, so we can do either this:

```
if (item == null)    // this robot flew off sick
{
}
```

or this:

```
if (item != null)    // this robot reported for duty
{
}
```

Algorithm
You need to put all this together …

- count up how many are sick and how many have reported for duty.
- Add 1 to appropriate counters within the loop.
- Use **message()** instructions to show your counts at the end.

Here is a partial algorithm to help.

Algorithm (part solution)
1. Set counter to zero
2. **Loop 10 times**
   a. move forwards 5 metres.
   b. pause for 1 second
   c. use radar with limited range
   b. **if** NO WingedShooter detected
      i. add 1 to counter
   end if
   End Loop
3. Display counter value

To finish the task

- The robot's battery is very weak.
- There is a PowerStation nearby, so add some more code at the end of your program to do this :
  - Use your radar to find the PowerStation
  - Just go there and your robot will automatically start recharging

Extra
Notice that all the robots have names. Can you display the name of each fit robot in a suitable way as you count them (n.b. **item.name** has this information after the radar is used)

Example message: **Robot <*Brian*> Counted!**

Week 3: Independent Study (4 Tasks)

The following exercises will be marked. Attempt them outside of class, and copy your code, as well as screenshots, and algorithms into a logbook. In week 5 you will be required to submit this logbook electronically.

8 Ceebot Task 5.5: Draw Using Variables (a further modification)

You should already have a program that will draw rectangles of any size and length, inputting the lengths required. But there is a problem .. what happens if the length and width are too big? Try it .. the robot hits a barrier and blows up!

Your task

Modify your existing program (exercise 10 from week 1) so…

- the user is asked what length and width they require for the rectangle
- the program **only** draws a rectangle **if** the length **and** width are in the right range (maximum 23 metres) and a message is displayed saying:
  Rectangle of length <…> **metres and width** <…> **metres completed.**

- If the value input is wrong a suitable message is displayed:
  Length is too big
  or **Width is too big**

Testing

Create a new test plan for your program and test it using the following inputs:

Length	Width
30	30
20	20
30	20
20	30
23	24
24	23
24	24
23	23

Does the program always behave as you expected? This testing should reveal any problems at the **boundary** value (**23**). You may have to adjust your program as a result of this testing.

- You should put **algorithm**, **code** and completed **test plan** into your log book.
- Add a comment about the testing that you did and any program changes needed.

9

Ceebot Task 9.6: Alien Destruction

In front of you is a row of **TargetBots** and **AlienEggs**. After a few moments, the eggs will hatch into either **AlienSpiders** or **AlienAnts** and some of the ants can be very aggressive!

Your task

Use your WheeledShooter robot to get rid of the aliens without destroying the Targets. Each time you reset this exercise, the situation is different so you can't necessarily know where the aliens will be.

Program Guidance

1. There are 20 objects altogether (TargetBots and eggs) so it is sensible to use a **for loop** to repeat 20 times.
2. The objects are positioned **5 metres** apart, so you should program your robot to move this distance inside the loop.
3. You can use your radar to help you detect a **TargetBot** and then avoid all these positions.

Hint:

- In an earlier exercise of this unit (Roll-Call 1) you learned how to point your radar to the right ….

  item = radar(TargetBot, -90, 5);

  will use a narrow (5 degree) radar beam pointing to the right.
- Note that item will be **null** if nothing is detected.

Algorithm

One possible design for your program could be:

Algorithm

1. Loop 20 times
   a. use radar directed right to detect TargetBot
   b. if TargetBot not detected
      i. turn right
      ii. fire
      iii. turn back
   end if
   c. move 5 metres forward to next position
   end loop

Code this algorithm and get it to work.

Extra

There is another problem .. your PowerCell may not always last to enable you to destroy all the aliens.

- but there is a **PowerStation** nearby which will recharge your cell if you just go there.
- use your radar to get its position in the normal way.
- you should regularly check your PowerCell energy (**energyCell.energyLevel**) .. if this is below 0.3 you should go off and recharge it at the PowerStation.
- How can you then return to your previous position?

Put your algorithms and code into your logbook.

10 Ceebot Task 12.2: Testing 2

In the previous exercise (exercise 12.1 in **last week exercise 5**), power cells can sometimes run down completely during a test. You need to stop this from happening.

Your task:

- Modify the previous program (**you'll need to complete exercise 12.1 first**) so that it stops when the **energyLevel** reaches a low level (**0.2**). It should also stop, as before, when the required number of tests is done.
- So you will need to put **2 conditions** at the start of the **while loop** (see the note below)
- At the end of the test loop, you should **output** 2 more messages:
  - output **how many tests** were completed out of the number required
  - output a **message** saying whether the powercell **failed** or **passed** the tests (it fails **if** the number of tests completed is less than the number required to be done)

e.g.: **3 tests completed out of 5 Power Cell Failed Test** **OR** **4 tests completed out of 4 Power Cell Passed Test**

Note: You will need to use **&&** or **||** logical operators (which??) to **combine** 2 conditions.

11 Ceebot Task 25.1 : Nascar 1

For this task you have to program a racing car to drive round an oval track marked out with Barriers. Basically you need to prevent the car from hitting any barriers!

Some Hints

1. You need an infinite loop
2. Inside the loop, use the **drive()** instruction with 2 variables for the bot speed and bot direction
   e.g **drive(botspeed, botdirection);**
3. Use a brief **wait(0.01)** after the drive() instruction to allow some movement to take place
4. A botspeed of 1 gives maximum speed, 0 minimum
5. A botdirection 0 is straight ahead, 1 is maximum left and -1 is maximum right
6. Use **radar()** to detect a **Barrier** and change the botdirection if necessary
   e.g. **item = radar(Barrier, -30, 30, 0, 20);** // detects front right up to 20 metres away

Press the [F1] key to see more information and a possible algorithm
Note .. To start nascar programs you click a different button - bottom right.

Document your attempt (even if it's not finished) in your logbook.

If you have some success, you could try your code in the Nascar 2 and Nascar 3 exercises (Although this is optional – you won't be required to submit Nascar 2 or 3)
Hint: You may need to add code to avoid hitting other **WheeledRacer** robots!