Ceebot Programming

Programming Concepts CO452

Study Pack for Ceebot

Part A Weeks 1-3

Original work by Brian Ward

Contents

Learning Outcomes

for the Programming Concepts module: CO452

On successful completion of the module the student will be able to:

- Analyse a simple requirement in a structured manner in order to establish a strategy to solve the current problem
- Design, document, implement and test reliable, maintainable programs as solutions to simple problems
- Use structured techniques of design and implementation and good documentation practice

Make effective use of software development tools when implementing fit-for-purpose solutions

General Introduction

Welcome to Programming Concepts

- The CO452 module plan and method of assessment for this module is detailed at the back of this booklet, and you will also find the assessment criteria there.
  We want you to enjoy this module and achieve a good result. Therefore it is important that you read the module plan and assessment criteria at your leisure.

- You will need an electronic A4 logbook to record your work. Please get this up and running as soon as possible. **Classwork will be checked and marked** each week.

- You start this module by using Ceebot, a highly visual environment for learning fundamental programming concepts. It uses a C++/C#/Java programming style, so principles learnt here will transfer easily to other programming areas. Hopefully you will also find Ceebot fun to use. The first few exercises should be fairly easy, but you will find that they get more challenging in later weeks. Near the end of the module you will be introduced to the C# language .Good Luck!

- In the first introductory session you will learn the essentials of working with **Ceebot**. Then you will progress by using the most important principles of programming.

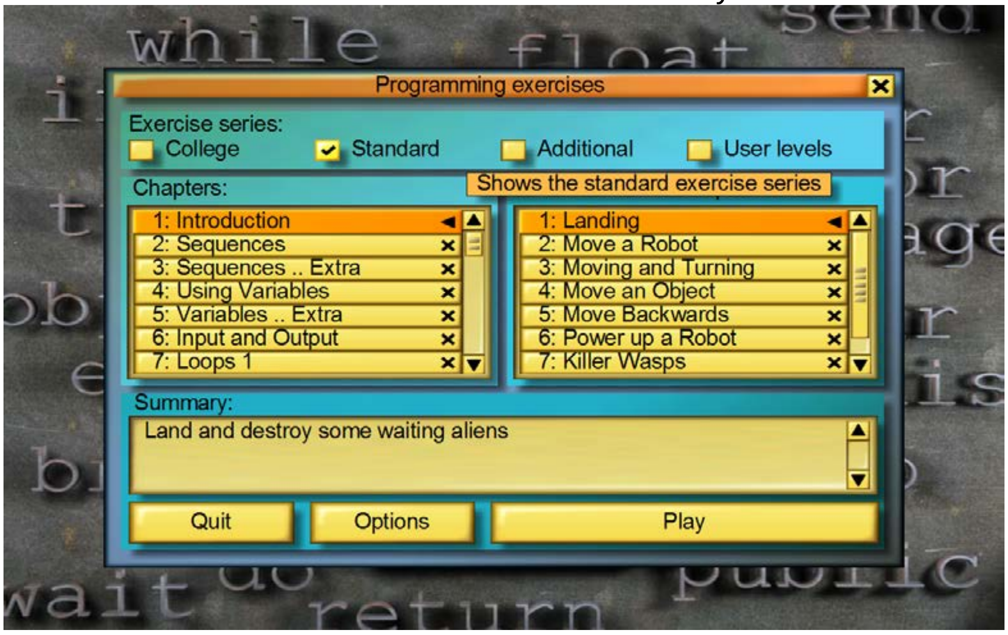
- Please Note: There are hundreds of exercises in the Ceebot package. **You are NOT expected to complete them ALL!** The exercises that you need to complete will be explained here in this document. Of course you CAN do the others if you want to!

- For your convenience, the details of each task are summarised for you in this booklet, but you will also find information by pressing the **[F1]** key during an exercise. Using the **[F2]** key will bring up general support for the current chapter.
  - ➢ If you wish, you will be able to purchase a copy of Ceebot and the exercises for your own personal use (for a nominal charge).

Classwork

Remember, you do NOT have to do all the Ceebot exercises in the package. The ones you need to attempt are detailed here in this booklet.

You must use the Standard set of exercises. Try others afterwards if you want to.



The first thing to keep in mind is that when you have selected an exercise and it has loaded:

- the **[F1]** key will always bring up the instructions for the current exercise.
- the **[F2]** key will always give you more general help with the chapter and the instructions you may need to use to complete a task..

This week you are to try to complete some tasks in class from the first few **Ceebot** chapters: Ask for help if you need it.

- Show your solutions to your lecturer as you complete them
- Include **comments** in your code where relevant (ask your lecturer how to do this)
- You should include the following information in comments at the top of each program:
  // Programmer's name: and ID:
  // Course:
  // Week No: and Exercise No:
  // Date:
- Copy your finished code and put this into your logbook with appropriate **headings** (Note .. you can cut and paste into MS Word or WordPad for later printing)

Your Log Book

You need to include the Independent study tasks in your logbook for electronic submission. These exercises will be marked.

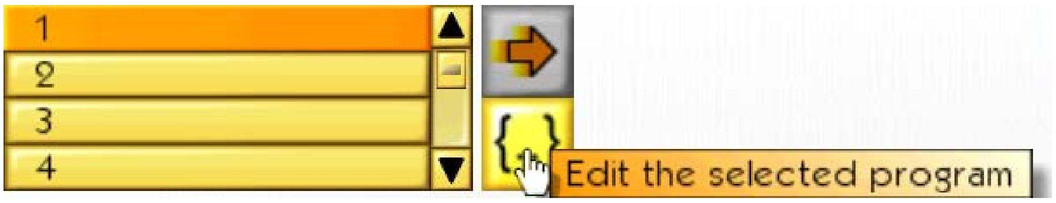
- Put **Unit Headings** and **Task Headings**.
- Give a brief description of the task
- Stick in your commented source code solution
- You may sometimes need other documentation such as algorithms or test plans.
- Add brief comments as to your success or otherwise and any problems that occurred

An example of how Ceebot works

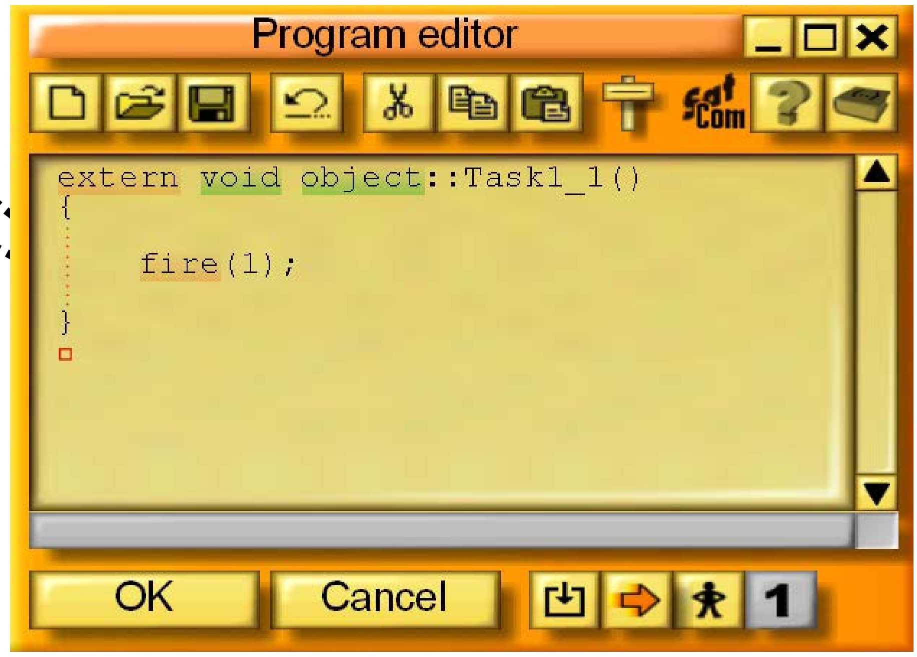
Ceebot Task 1.1: Landing

This exercise starts with a movie showing your spaceship landing on a hostile alien world. Once you have landed, you will see a robot shooter in the distance on the planet surface and 2 alien ants that threaten to destroy this robot.

- Your task is to program the robot to destroy the ants using the **fire(…);** instruction.
- First **Click** on the **robot shooter** in the distance, then select its **editor** (see below)



- The editor allows you to enter programs and compile them
- Type the instruction to fire the robot cannon for 1 second
- Then **compile** the program to see if there are any errors
- If errors exist, correct and try again

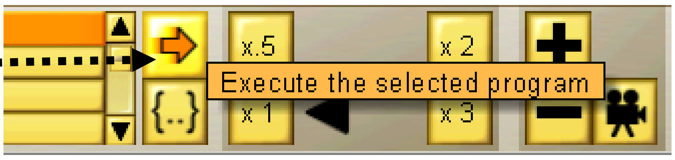


```
extern void object::Task1_1()
{
    fire(1);
}
```

Destroying Both Ants

- You will notice that there are **2** ants! So you must write the fire instruction twice
- Your robot should also turn round through 180 degrees before the second firing.
- There is a **turn(..);** instruction that should help.
- Can you put **3** instructions together in **one** program to finish the exercise?
- Click **OK** when you have done this

Executing the Program



- You now have to **run** (execute) this program by clicking the arrow button
- If you leave it too long, the robot will be destroyed, but don't worry .. you can always **restart** the exercise
- remember you can get more help by using the **[F1]** and **[F2]** keys

Important Notes (saving and restarting)

- When you click OK in the editor, your program is **saved** automatically .. so if you restart it, your old program should not be lost!
- When you restart, you can skip the introductory movie by using the **[Esc]** key
- You can restart any exercise as many times as you like

Week 1

Sequence, Variables, Input and Output Class exercises

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

Your task

Program the **WheeledGrabber** to reach the platform that you can see in the distance. The path to take is marked out with blue waypoints (checkpoints). These are all **20 metres** apart.

- Use the **editor** to create the program.
- You will need to use the **move(…)** and **turn(…)** instructions.
- Notice that most programming instructions have brackets. These are called functions. Inside the brackets you can put a value (called a **parameter**) and this value is then passed to the function. 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** (quicker to change values), so start by setting up some variables for the length and angle:

```
int  len;          // length
int  angle;
```

Algorithm

```
1. Set length to ??
2. Set angle to ??
3. Move forward using length
4. Turn using angle
5. Move forward using length

etc.
```

Note: The **turn()** function will only accept an integer parameter. It uses the angle **anti-clockwise** so **90** will turn **right**; **-90** will turn **left**.

2 Task 5.3: RoboMaths

Your Task

Implement the algorithm displayed below:

- 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 …

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.

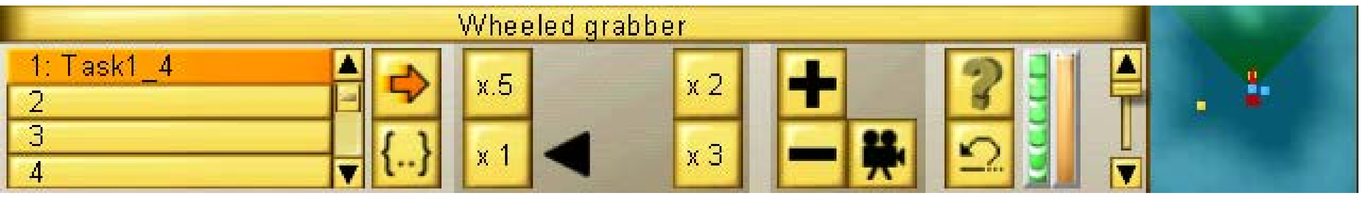
Your task is to reach the platform and survive, by asking the user to **input** a distance to move, then to move the robot through that distance. If you keep running the program, you should eventually reach your destination.

- 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**

4 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 the **WingedGrabber** using the **WheeledGrabber** ..

- Click on 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.

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?

6 Ceebot Task 6.5: Target Practice

Your task

- Write a program to input the horizontal and vertical angles and then use these to destroy one of the explosive 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:

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)

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

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	

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 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

7 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.

8 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. 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

9 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

10 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

Week 2

Iteration

There is often a need to **repeat** programs or parts of a program. This is known as **iteration** (another word for **looping**). This week we shall look at 3 methods of looping:

The for loop

The for loop used when we know how many times we want to repeat a block of statements. There are 3 parts to the statement: initialisation, condition and increment. The following will repeat **10 times:**

```
int count ;            // declare loop counter

for (count=0; count < 10; count++)
{
      // ---- put instructions to be repeated here
}
```

The while loop

The while loop continues to repeat a **block** of statements while some **condition** in brackets remains **true**. The while loop uses a **variable** as a **loop counter**, keeping a count of how many loops have been done. The following repeats **10 times:**

```
int count = 0;          // initialise loop counter to zero

while (count < 10)      // continue while count less than 10
{
            // ---- put instructions to be repeated here

        count++;        // add 1 to loop counter
}
```

The do while loop

The **do .. while … loop** is different from the **while loop** as it always executes the block of instructions at least **once** because the condition is checked at the **end** of the loop.

```
do
{
        // ---- put instructions to be repeated here
}
while (condition);    // while the condition is true
```

1 Ceebot Task 7.6: Calculator 2

Your task is to use a **for loop** to input 4 numbers. When the loop has finished, output the total of the 4 numbers

- You will need to use the **dialog(…)** and **strval(…)** instructions as well as the **message(…)** instruction to output to the screen.
- And, of course, you will need a **for loop**.
- Note: your input prompts should look like this:
  Enter Number 1
  Enter Number 2
  etc.

Testing
Design a suitable **test plan** for your program, with at least 3 different tests.

Extra
Can you also display the **average** (of the numbers entered) in the same message as the total?

2 Ceebot Task 7.2: Massacre

Your task is to destroy the 10 spiders that surround you.

- As there are 10 spiders you need a loop that repeats 10 times.
- Before you start, you will need to set up the variables you need.

- A suitable **algorithm** for the program is:

- You should note that the spiders have conveniently arranged themselves **15 degrees apart**.

Algorithm
```
 1. Loop  10 times
          a. fire cannon for 0.1 seconds
          b. turn to next spider
          c. pause for 1.5 seconds
    End Loop
```

Extra
Now also get the program to display a message after each firing:
"Spider 1 Destroyed" etc.

3 Ceebot Task 11.3: Shooting Practice

Your task: Using **one** series of statements, destroy **all** of the targets

Hint
Using **nested loops** is the most efficient way. Create a loop (the inner loop) that destroys one side of targets, and then repeat that loop 4 times (the outer loop) turning after each side is destroyed.

4 Ceebot Task 7.3: Blasted Ants

This exercise uses an **infinite loop** to destroy all the attacking ants. First set up a **while loop** with **true** condition .. this makes the loop continue forever!

- Inside the loop you need to keep firing as you turn your robot. Try using **fire()** and **turn()** .. it doesn't work. You can turn or fire, but not both at the same time!
- What you need is a new instruction: **drive(..)** .. which has 2 parameters, the **speed** and the **direction** of turning (both can vary from -1 to +1)
- **drive(0, 1);** will starts the robot turning on the spot in an anticlockwise direction .. other instructions such as **fire(..);** will still work as the robot turns.
- Put all this together to create your program

Extra: Can you do another program that uses your radar to detect the ants before firing?

5 Ceebot Task 12.1: Testing, Testing!

Power Cells are not what they used to be! You have been asked to design a test program for them that can be run in a loop **any** number of times.

Your task:
Put together a sequence of robot instructions to run in a loop. Your test sequence should have a few movements, turns and firing. An example is given in the algorithm below. The test will use up energy from the power cell and you can then display this energy level (how?... see the note below)

- Start the program by asking the user to **input** how many tests are to be run
- then use a **while loop** to run that number of tests.
- You should **output** messages at the beginning and end of each test:
  Starting Test 1 (etc.) and
  Finished Test 1: Energy Level = …… (etc.)

A partial **algorithm** for the program is:

Note: **energyCell.energyLevel** gives the current level of the power cell.

When the program is working, **test both robots** to see which uses most energy for the same number of tests.

Algorithm (partial)

```
1.  Set counter to zero
2.  Input the number of tests to run
3.  Loop while counter < number of tests
        a. output starting test message
            ▪ move forward 3 metres
            ▪ turn 180 degrees
            ▪ fire for 0.5 seconds
            ▪ move back 3 metres
        b. output finish test message
        c. output current energy level of power cell
        d. add 1 to counter
    End Loop
```

6

Ceebot Task 8.4: Exchange Posts 2

This exercise uses **Information Exchange Posts** which store information that can be picked up by any robot within 10 metres. Your robot is on a very dangerous path through a lava lake. You may just be able to see your goal on the right .. a tall **Lightning Conductor** in the distance across the lake.

Your task .. reach the goal destination without falling into the lake!
But how can you know what direction to take and how much to move your robot?

- This information is stored in each of the 9 **Exchange Posts**.
- If you click on one of these you will see that it stores a **Number**, a **Direction** and a **Length**. Each Exchange Post tells you how to get to the next one.
- There are 9 Exchange Posts altogether so use an appropriate **for loop** for this.

Program Guidance
What must you do inside the loop?

- You need to **receive** the Direction and Length information from an Exchange post.
- Then use **turn(…)** and **move(…)** with these received values.

How do you receive the information?.

- First set up two float variables to hold the information ..

```
float  dir;      // declare variable to hold direction
float  dist;     // declare variable to hold distance
```

- then use the **receive(…)** instruction twice …

```
dir = receive("Direction");   // pick up Direction info from Exchange Post
dist = receive("Length");    // pick up Length info from Exchange Post
```

Design an algorithm for the program. Then use it to produce a working program

Note: Speed up your program using **x2** or **x3** buttons. Fly the astronaut to watch the action.

Week 2: 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.

7 Ceebot Task 8.1: Flying Shooters

In this scene there are 10 **WingedShooter** robots .. troops that are programmed ready to fly off to attack the large nest of **ants** in the distance. If you look at the **minimap** at the bottom right hand corner of the screen, you will see stars representing all the ants.

Your task
equip all robot troops with **PowerCells** so they can fly off to get the job done.

- As there are 10 robots, you should use a **while loop** to repeat your actions 10 times, although you may not need all of the shooters.
- Although the WingedShooters are waiting patiently in a line, the PowerCells are scattered around the scene. You will have to use your **radar** to get the position of each **PowerCell**.
- You can then use the radar again to find the nearest **WingedShooter** to put it in.
- Here is an **algorithm** to help you:

Now translate this algorithm into a working program

Note: If you wish, you can click on the other robot at the top left of the screen and watch the action at the nest!

Algorithm

```
1. Set count to zero
2. Loop while count less than 10
      a. Use radar to get PowerCell details
      b. go to this position
      c. pick up Power Cell
      d. Use radar to get WingedShooter details
      e. go to this position
      f. drop Power Cell
      g. add 1 to count
   end Loop
```

8 Ceebot Task 8.6: Loop the loop

Your task: Using a nested loop, draw 3 squares which increase in size to pass through the Checkpoints. The lengths of the 3 squares will be 10, 15 and 20 meters. Draw the squares in the colour blue.

Hint
You'll need to modify the length of which you move in the inner loop, after a square is drawn. It would be a good idea to set up a variable to store and refer to the modified length.

9

Ceebot Task 15.1: Zig-Zag

Your Task

You have to reach the **GoalArea** at the end of the path without being destroyed by the mines along the way. You will see that the path you must take is marked out for you.

Hints

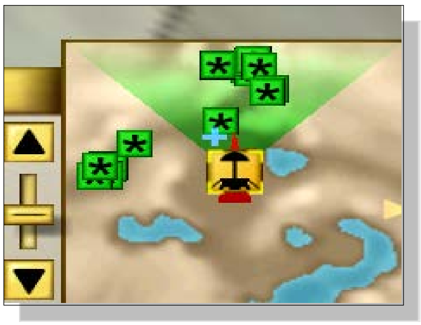
- First turn 45 degrees to face in the right direction
- Then use a **do while** loop that continues until you reach the GoalArea.
    - o You can use your radar to detect the **GoalArea**
    - o then work out the distance to it.
- Continue the loop until you are **just 1 metre away** from the GoalArea.
- You will need to move in 7 metre steps
- You will need to turn 90 degrees at the end of each step (but not in the same direction!)

10

Ceebot Task 12.7: Ant Attack

Your Task is to use the **LeggedShooter** to destroy the **AlienAnts**. You can see them marked on your **mini-map** at the bottom right corner of the screen.

Program Guidance



- The ants will attack from all angles .. but there is a blue WayPoint in the distance marking a good place to position your robot and make your stand.
    - o So start by using your **radar** to detect the **WayPoint** and go to it.
- Then you will need to use your **radar** again to detect an **AlienAnt** and then turn and fire to destroy it. Remember that **null** will be returned if nothing is detected.
- As there are many ants, you will need to do this in an **indefinite loop**
- Here is an **algorithm** to help you:

Algorithm

```
1. Use radar to detect WayPoint
2. Go to the WayPoint
3. Loop indefinitely
      a. Use radar to detect AlienAnt
      b. if Ant detected
            i. turn towards the Ant
            ii. fire
            iii. pause briefly
   End Loop
```

Extras

1. You may find that you could run out of ammunition by firing when the ant is too far away. Alter the radar instruction to detect only ants within, say, 40 metres.
2. You are wasting ammunition by firing when the ant is too high. Find a way to fire only when the ant is about level

Hints for 2

- the **z coordinate** is the height above sea level
- **this.position.z** is the z coordinate of the robot
- **item.position.z** is the z coordinate of an object (item) detected by radar.

(note: altitude can't be used here because it is the height above ground: zero in this case)

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:

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

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

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!

Appendix A: Examples of Important Ceebot Instructions

1. Ceebot Specific

move (20);	.. move 20 metres
turn(180);	.. turn 180 degrees anticlockwise
wait(1.5);	.. wait for 1.5 seconds
fire(2.5);	.. fire cannon for 2.5 seconds
aim(10);	.. aim cannon 10 degrees up (use values from -20 to +20)
grab();	.. grab the item directly in front
drop();	.. drop the item being carried
pendown();	.. put the pen against the floor ready for drawing
penup();	.. lift the pen to stop drawing
red();	.. select a red pen for drawing (various colours available)
drive(1, 0);	.. drive forward at full speed
jet(1);	.. fly upwards at full speed (use values -1 to +1)

2. Input and Output

name = dialog("Enter Name");	.. show dialog to input name and store it in name variable
message("I am " + name);	.. output a message with text joined to a name variable
num = strval (dialog("Enter number"));	.. enter string and convert to its number value
item = radar(WheeledShooter);	.. get details of the nearest WheeledShooter robot
keypushed(VK_UP);	.. detects pressing of a key (e.g the UP arrow key) .. used with if()

3. Variables

int count;	.. define a variable called count to store an integer number
float num;	.. define a variable called num to store a float (decimal) number
string name;	.. define a variable called name to store a string (text or words)
object item;	.. define a variable called item to store an object's details
point here;	.. define a variable called here to store a position (x and y)

4. Assignments to Variables (must be defined first)

count = 0;	.. put 0 into the count variable (previously defined as int)
num = 5.67;	.. put 5.67 into the num variable (previously defined as float)
name = "Fred Bloggs";	.. put these 11 characters in the name variable (defined as string)
here = this.position;	.. store current position of robot in the here variable (a point)
angle = direction(item.position);	.. find angle of item from you (after using radar)
dist = distance (item.position, this.position);	.. find distance from item to your position.

5. Calculations

count ++;	.. add 1 to the value of the count variable
count --;	.. subtract 1 from the value of the count variable
count = count + 3;	.. add 3 to value of the count variable (or use **count += 3;**)
count = count - 6;	.. subtract 6 from the count variable (or use **count -= 6;**)
av = (num1 + num2 + num3 + num4) / 4;	.. work our average of 4 numbers
tax = bill * 17.5 / 100;	.. work out 17.5 percent tax on your bill

6. Loops (iteration)

a. The while loop

```
int  count = 0;      // initialise a loop counter to zero

while (count < 10)     // continue while loop counter is less than 10
{
        message ("The count is " + count);   // repeated message
        count ++;       // keep loop going by adding 1 to counter
}
```

b. The for loop

```
// initialise loop counter; continue while count less than 10 ;  add 1 at end of loop

for (int count = 0; count < 10; count ++)
{
        message (“The count is " + count);   // repeated message (10 times)
}
```

c. The do while loop

```
int  count = 0;      // initialise a loop counter to zero

do
{
        count ++;       // keep loop going by adding 1 to loop counter
          message (“The count is " + count);   // repeated message
}
while (count < 10);     // continue while loop counter is less than 10
```

7. Selection

a. The if statement

```
int  count = 0;

while (count < 10)
{
        if (count == 4)  // if count is equal to 4
        {
                message (“We are half way" );
        }

        count ++;
}
```

b. The if else statement

```
        if (count >= 4)  // if count is greater or equal to 4
        {
                message (“We have reached half way" );
        }
        else
        {
                message ("We are NOT half way yet");
        }
```

c. The switch statement

```
switch(count)  // use count value to switch to various cases below:
{
        case 1:                          // i.e. if  count value = 1
                message (“We are just starting" );      break;
        case 2:  case 3: case 4:
                message (“We are on our way" );         break;
        case 4:
                message (“We are half way" );           break;
        default:
                // do nothing for any other values
}
```

8. Conditions

(a == b) .. a is equal to b ?
(a > b) .. a is greater than b ?
(a < b) .. a is less than b ?
(a >= b) .. a is greater or equal to b ?
(a <= b) .. a is less than or equal to b ?
(a != b) .. a is NOT equal to b ?

(a == b **||** a == c**)** .. a is equal to b **OR** a is equal to c ?
(a == b **||** a == c **||** a == d**)** .. a is equal to b **OR** a is equal to c **OR** a is equal to d ?

(a == b **&&** a == c**)** .. a is equal to b **AND** a is equal to c ?
(a <= 100 **&&** a >= 0**)** .. a is less than or equal to 100 **AND** a is greater or equal to 0 ?

9. Functions

```
void  object::myFunc()
{
    message ("I am now inside the myFunc function");
}
```

*// this defines a function called **myFunc** which has no parameters and returns nothing (void)*
*// to use it, you 'call' it using its name : i.e. **myFunc();***

10. Functions with parameters

```
float  object::Tax(float  amount)
{
     float  taxAmount;                 // local variable
      taxAmount = amount * 17.5/100;
      return  taxAmount;
}
```

*// this defines a function called **Tax** which has 1 parameter and returns a float value*
*// to use it, you can 'call' it like this: **vat = Tax(Bill);***
// this passes the Bill value into the function and picks up the returned tax value from it.
([F2] key for more)

Appendix B: The Basics of C# (Console)

1. Input and Output

name = Console.ReadLine(); .. store input in a name variable (defined as string)
Console.WriteLine("I am " + name); .. output a message with text joined to a name variable
num1 = Convert.ToDouble (Console.ReadLine()); .. enter string and convert to a double
num2 = Convert.ToInt32 (Console.ReadLine()); .. enter string and convert to an integer

2. Variables

int count;	.. define a variable called count to store an integer number
double num;	.. define a variable called num to store a double (decimal) number
string name;	.. define a variable called name to store a string (text or words)

3. Assignments to Variables (must be defined first)

count = 0;	.. put 0 into the count variable (previously defined as int)
num = 5.67;	.. put 5.67 into the num variable (previously defined as double)
name = "Fred Bloggs";	.. put these 11 characters in the name variable (defined as string)

4. Calculations

count ++;	.. add 1 to the value of the count variable
count --;	.. subtract 1 from the value of the count variable
count = count + 3;	.. add 3 to value of the count variable (or use **count += 3;**)
count = count - 6;	.. subtract 6 from the count variable (or use **count -= 6;**)
av = (num1 + num2 + num3 + num4) / 4;	.. work our average of 4 numbers
tax = bill * 17.5 / 100;	.. work out 17.5 percent tax on your bill

5. Loops (iteration)

a. The while loop

```
int  count = 0;      // initialise a loop counter to zero

while (count < 10)     // continue while loop counter is less than 10
{
       Console.WriteLIne (“The count is " + count);   // repeated
       count ++;       // keep loop going by adding 1 to counter
}
```

an infinite loop

```
while (true)     // continue the while loop forever
{
       Console.WriteLIne ("Yippeeee!!");   // repeated forever
}
```

b. The for loop

```
// initialise loop counter; continue while count less than 10 ;  add 1 at end of loop

for (int count = 0; count < 10; count ++)
{
      Console.WriteLine (“The count is " + count);   // repeated 10 times
}
```

c. The do while loop

```
int  count = 0;      // initialise a loop counter to zero

do
{
        count ++;       // keep loop going by adding 1 to loop counter
        Console.WriteLine (“The count is " + count);   // repeated message
}
while (count < 10);     // continue while loop counter is less than 10
```

6. Selection

a. The if statement

```
      if (count == 4)  // if count is equal to 4
      {
            Console.WriteLine (“We are half way" );
      }
```

b. The if else statement

```
      if (count >= 4)  // if count is greater or equal to 4
      {
            Console.WriteLine (“We have reached half way" );
      }
      else
      {
            Console.WriteLine ("We are NOT half way yet");
      }
```

c. The switch statement

```
switch(count)  // use count value to switch to various cases below:
{
        case 1:                              // i.e. if  count value = 1
                Console.WriteLine (“We are just starting" );            break;
        case 2:  case 3: case 4:
                Console.WriteLine (“We are on our way" );               break;
        case 4:
                Console.WriteLine (“We are half way" );          break;
        default:
                // do nothing for any other values               break;
}
```

7. Conditions

(a == b) .. a is equal to b ?
(a > b) .. a is greater than b ?
(a < b) .. a is less than b ?
(a >= b) .. a is greater or equal to b ?
(a <= b) .. a is less than or equal to b ?
(a != b) .. a is NOT equal to b ?

8. Multiple Conditions

(a == b **||** a == c**)** .. a is equal to b **OR** a is equal to c ?
(a == b **||** a == c **||** a == d**)** .. a is equal to b **OR** a is equal to c **OR** a is equal to d ?

(a == b **&&** a == c**)** .. a is equal to b **AND** a is equal to c ?
(a <= 100 **&&** a >= 0**)** .. a is less than or equal to 100 **AND** a is greater or equal to 0 ?

9. Classes, Objects and Methods

```
class Meal                    // define a class called Meal
{
        private string food;    // the class has one class variable (attribute or field)

           public static void Main()         // program starts executing here
           {
                 Meal myMeal = new Meal();           // create a new myMeal object
              myMeal.getFood();                // call the object's getFood() method
        }

        public Meal()                        // this is the Meal class constructor
        {
                food = "Fish and Chips";    // this sets the default food
        }

        public void getFood()            // define a method getFood()which returns nothing
(void)
        {
                Console.WriteLine("What would you like to eat?");
                food = Console.ReadLine();             // input into the class variable food
        }
}
```

*// this defines a simple class called **Meal** which has one variable, one method, one constructor*

10. Methods with parameters

```
public double setTax(double  amount)
{
    double  taxAmount;           // local variable
      taxAmount = amount * 17.5/100;
      return  taxAmount;
}
```

*// this defines the method **setTax()** which has 1 parameter (amount) and returns a double value*
// this method will be defined inside a class e.g the Meal class above
// to use it, you can 'call' it like this:

```
                vat  =  myMeal.setTax(Bill);      // assume myMeal is the object created from
        Meal
```

// this passes the value of Bill into the method and picks up the returned tax value from it.

Assessment of CO452 Programming Concepts

1. This module is assessed by coursework. There are three parts to this coursework (Part A, B and C). There are study packs for each of the three parts. The study packs contain both class exercises and independent exercises relating to the programming concept being taught that week. There is a project week in Part B that includes a series of related tasks.

2. **Class exercises will be assessed**. Each week contains between four to six class exercises. Your tutor will monitor your progress in these each week. **These class exercises are worth 40% of your Part A mark.**

3. **Independent studies (and project tasks in Part B) will be assessed.** The code for these tasks will be assessed on their efficiency, syntax, correct use of concept, and whether the code fulfils the requirements of the task. Some tasks may also require additional documentation such as test plans and algorithms. Please include screenshots of your code running and comments where relevant**. You must complete these independent exercises on your own outside of the session. These exercises are worth 60% of your Part A mark.**

4. Create a logbook (for example: an MS Word document) to document your code. The logbook should contain your designs, algorithms, test plans, source code and results of your work. **This must be submitted electronically through the designated TurnItIn submission point** (your tutor will show you). **If there is a technical problem and you cannot submit through TurnItIn, please speak to someone from the administration office (E4.08).**

5. Your mark for this module will be based on your grades for each of the parts (A, B, C). Below shows the weighting for each part of the coursework:

Part A: 30% of module mark

Week 1, 2 and 3 class exercises = 40% of Part A mark

Week 1, 2 and 3 independent exercises = 60% of Part A mark

Part B: 40% of module mark

Week 5 (TBD), Week 6 (TBD), Week 7 (TBD), Week 8 (Project)

Part C: 30% of module mark

Week 10 (TBD), Week 11 (TBD), Week 12 (TBD)

Grade related criteria for Programming - CO452

A	Where the student has demonstrated clear evidence of an excellent understanding of the theories and principles together with a high degree of analytical accuracy, good design skills, implementing fully tested solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar sessions and attempt at least 85% of independent studies.*
B	Where the student has demonstrated clear evidence of a good understanding of the theories and principles together with a good analytical ability, good design skills, implementing solutions that show reliability, maintainability, readability and minimal complexity and correct form of presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar sessions and attempt at least 75% the independent studies.*
C	Where the student has demonstrated a reasonable understanding of the theories and principles together with a reasonable analytical ability, design skills, implementing solutions that appreciate the need for reliability, maintainability, readability and minimal complexity and reasonable presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar session and attempt at least 66% of the independent studies.*
D	Where the student has demonstrated an understanding of the theories and principles of analysis, design, implementation and presentation skills. *To acquire the knowledge and skills to demonstrate the above the student will normally be expected to attend the seminar session and attempt at least 50% of the independent studies.*
E	Where the student has made a genuine attempt to acquire the knowledge and skills but requires further application and study to demonstrate an understanding of the theories and principles of analysis, design, implementation and presentation skills. *In order to demonstrate a genuine attempt the student will normally be expected to attend the seminar sessions and attempt at least 40% of the independent studies.*
F	Where the student has clearly not acquired sufficient knowledge and skills and not attempted or coped with the directed study with any degree of competence regarding theories, principles, analysis, design, implementation and presentation skills or where the student has NOT attended for assessment or where the student has copied work from an alternative source.

Module Name and code	**Programming Concepts CO452**
Staff: Carlo Lusuardi, Richard Jones, Nick Day, Based on original work by Brian Ward	

Learning Outcomes:
- Analyse a simple requirement in a structured manner
- Design, document, implement and test reliable, maintainable programs as solutions to simple problems
- Use structured techniques of design and implementation and good documentation practice.
- Use software development tools.

WK		LECTURE/TUTORIAL	PRACTICAL
1		**INTRO to Ceebot, VARIABLES, INPUT and OUTPUT**	Ceebot Chapters 1-6
2		**ITERATION**	Ceebot Chapters 7-8; 12-15
3		**SELECTION**	Ceebot Chapters 9-10
4		**WORKSHOP for CW 1 Part A submission next week**	
5		**FUNCTIONS**	Ceebot Chapters 18-19
6		**PARAMETERS**	Ceebot Chapters 20-21
7		**ARRAYS**	Ceebot Chapters 22-23
8		**Ceebot PROJECT**	Ceebot Chapter 24
9		**WORKSHOP for CW 1 Part B submission next week**	
10		**C# 1 Input and Output**	C# Intro Directed Study Pack: Unit 1
11		**C# 2 Sequence, Selection, Iteration**	C# Intro Directed Study Pack: Unit 2
12		**C# 3 Classes, Objects and Methods**	C# Intro Directed Study Pack: Unit 3
		Christmas Break	
16(13)		**WORKSHOP for CW 1 Part C submission next week**	
17(14)		Review / Module Surgery	

Note: Weeks in () are Teaching weeks

Course Texts:
Comprehensive Course Notes are provided

Bradley & Millspaugh, ***Programming in C#***, 2010, pub: McGraw Hill
Deitel & Deitel, ***Visual C# 2010 How to Program***, 2011, pub: Pearson