Revision

Revision starts now!

1. Convert 21 into binary
2. Convert 0111 1101 into decimal
3. Convert 63 into binary, giving your answer as one byte

Extension: Convert 45, 605, 211 into binary using your calculator

Revision

Convert 21 into binary

Revision

Convert 21 into binary

10101

Revision

Convert 0111 1101 into decimal

Revision

Convert 0111 1101 into decimal

64 + 32 + 16 + 8 + 4 + 1 = 125

Revision

Convert 63 into binary, giving your answer as one byte

Revision

Convert 63 into binary, giving your answer as one byte

0011 1111

Revision

Convert $45,605,211$ into binary using your calculator

Revision

Convert $45,605,211$ into binary using your calculator

0010 1011 0111 1110 0001 0101 1011

SA 1

- First ‘subject assignment’
- One assessment each half term
- The first work is auto-marked
- Complete it independently alongside the **next** weekly assignment (A103)

Dry Runs

Specification

4.4.1.2 Following and writing algorithms

Content	Additional information
Understand the term algorithm.	A sequence of steps that can be followed to complete a task and that always terminates.
Be able to express the solution to a simple problem as an algorithm using pseudo-code, with the standard constructs: • sequence • assignment • selection • iteration.	
Be able to hand-trace algorithms.	

Tracing algorithms

- What is an algorithm?

Tracing algorithms

- What is an algorithm?
  - A sequence of steps that can be followed to complete a task and that always terminates

Tracing algorithms

- What is an algorithm?
  - A sequence of steps that can be followed to complete a task and that always terminates
- Tracing an algorithm is manually executing the algorithm
  - To identify its behaviour
  - To identify any potential errors
- It’s a useful skill for debugging – and you need to be able to do it for the exam

Example 1

```
read(x, y, z)
if x > y then
  if x > z then
    print x
  else
    print z
  end if
else
  if y > z then
    print y
  else
    print z
  end if
end if
```

What value is printed in each of the following cases?

1. $x = 5, y = 2, z = 3$
   •
2. $x = 1, y = 2, z = 4$
   •

What is the behaviour of this algorithm? What does it do?

Example 1

```
read(x, y, z)
if x > y then
  if x > z then
    print x
  else
    print z
  end if
else
  if y > z then
    print y
  else
    print z
  end if
end if
```

What value is printed in each of the following cases?

1. $x = 5, y = 2, z = 3$
   - 5
2. $x = 1, y = 2, z = 4$
   -

What is the behaviour of this algorithm? What does it do?

Example 1

```
read(x, y, z)
if x > y then
  if x > z then
    print x
  else
    print z
  end if
else
  if y > z then
    print y
  else
    print z
  end if
end if
```

What value is printed in each of the following cases?

1. $x = 5, y = 2, z = 3$
   - 5
2. $x = 1, y = 2, z = 4$
   - 4

What is the behaviour of this algorithm? What does it do?

Example 2

```
If X > 0 OR Y = 5 Then
  Display "Yes"
Else
  Display "No"
End If
If X > 0 AND Y = 5 Then
  Display "Black"
Else
  Display "White"
End If
If Not(X > 0 AND Y = 5) Then
  Display "Black"
Else
  Display "White"
End If
```

Assume $X = 1, Y = 1$. What is displayed?

Example 2

```
If X > 0 OR Y = 5 Then
  Display "Yes"
Else
  Display "No"
End If
If X > 0 AND Y = 5 Then
  Display "Black"
Else
  Display "White"
End If
If Not(X > 0 AND Y = 5) Then
  Display "Black"
Else
  Display "White"
End If
```

Assume $X = 1, Y = 1$. What is displayed?

```
Yes White Black
```

Suspects

- Professor Plum
- Reverend Green
- Miss Scarlet
- Mrs Peacock
- Col Mustard

Doctor Black has been found murdered downstairs in the kitchen, battered by some lead piping, at 20:00. Find out whodunnit by looking at the pseudocode.

The pseudocode is available on Google Classroom.

The murderer was...

The murderer was...

Reverend Green

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0		

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Total	Count	x
0	0	

Input for x:

- 1
- 2
- 3
- -1

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0	0	1

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0	0	1
1		

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Total	Count	x
0	0	1
1	1	

Input for x:

- 1
- 2
- 3
- -1

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0	0	1
1	1	2

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Print: ???

Total	Count	x
0	0	1
1	1	2
3		

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0	0	1
1	1	2
3	2	

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0	0	1
1	1	2
3	2	3

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Total	Count	x
0	0	1
1	1	2
3	2	3
6		

Input for x:

- 1
- 2
- 3
- -1

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Total	Count	x
0	0	1
1	1	2
3	2	3
6	3	

Input for x:

- 1
- 2
- 3
- -1

Print: ???

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Print: ???

Total	Count	x
0	0	1
1	1	2
3	2	3
6	3	-1

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Input for x:

- 1
- 2
- 3
- -1

Total	Count	x
0	0	1
1	1	2
3	2	3
6	3	-1

Print: 2

Exercise 1

```
Total ← 0
Count ← 0
read(x)
while x != -1
  Total ← Total + x
  Count ← Count + 1
  read(x)
end while
print Total / Count
```

Total	Count	x
0	0	1
1	1	2
3	2	3
6	3	-1

Input for x:

- 1
- 2
- 3
- -1

Print: 2

What does this code do?

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4
20	

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4
20	3

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4
20	3
60	

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4
20	3
60	2

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4
20	3
60	2
120	

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: ???

Num	Count
5	4
20	3
60	2
120	1

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: 120

Num	Count
5	4
20	3
60	2
120	1

Exercise 2

```
Begin
  Input(num)
  count ← num - 1
  While count > 1 do
    num ← num * count
    count ← count - 1
  Endwhile
  Display(num)
End
```

Input for num:

5

Display: 120

Num	Count
5	4
20	3
60	2
120	1

What does this code do?

Worksheet

L108 – Dry Runs Worksheet 1

This is on Google Classroom.

Iteration

Iteration

- So far, we have seen the concept of **looping**
- Formally, this is known as **iteration**

Iteration

- So far, we have seen the concept of **looping**
- Formally, this is known as **iteration**
- There are 3 types of loop:
  - **Definite** (fixed)
  - **Indefinite** (conditional)
  - **Infinite**

Definite loops

- A loop is **definite** when the number of iterations is determined before it starts
- Hence, the loop runs a **fixed number of times**

```
for (int i = 0; i < 100; i++)
{

}
```

Indefinite loops

- A loop is **indefinite** when the number of iterations is not explicitly defined before the loop starts
- The loop continues until or while a condition expression is true (e.g. `x > 0`)

```
string input = Console.ReadLine();
while (input != "exit")
{
  input = Console.ReadLine();
}
```

`while` loops in C#

- In C#, there are two ways we can write `while` loops

```
while (x > 0)
{
  x -= 1;
}
```

```
do
{
  x -= 1;
} while (x > 0);
```

- Is there any difference in the behaviour of these loops?

`while` loops in C#

- In C#, there are two ways we can write `while` loops

```
while (x > 0)
{
  x -= 1;
}
```

```
do
{
  x -= 1;
} while (x > 0);
```

- Is there any difference in the behaviour of these loops?
  - **Yes!** A `do-while` loop (on the right) will always run at least once. The `while` loop will only start if the condition is `true` initially.

Infinite loops

- A loop is **infinite** if it never stops
- Programs with infinite loops never **terminate**
  - This can be dangerous – be mindful of it

```
do
{
  Console.WriteLine("Hello!");
} while (true);
```

```
while (1 + 1 == 2)
{
  Console.WriteLine("Maths still works");
}
```

Additional loop terminology

- **`break`**
  - Immediately finish looping, regardless of whether or not the loop condition is met

```
int x = 2;
do
{
  x -= 1;
  if (x == 0)
  {
    break;
  }
} while (x > 0);
```

Additional loop terminology

- **`break`**
  - Immediately finish looping, regardless of whether or not the loop condition is met

```
int x = 2;
do
{
  x -= 1;
  if (x == 0)
  {
    break;
  }
} while (x > 0);
```

- **`continue`**
  - Immediately finish this iteration of the loop and start the next one (providing the condition is still met)

```
for (int i = 1; i < 20; i++)
{
  if (i < 10)
  {
    continue;
  }
  Console.WriteLine(i);
}
```

Summary

- Iteration (loops) is used when we want to repeatedly execute a **block** of code
- There are three types of loop:
  - Definite
  - Indefinite
  - Infinite
- **`break`** and **`continue`** are used to manipulate loops

Worksheet

W104 - C# `while`