

Starter

Evaluate the following RPN expressions on paper. Show your working!

1. 3 8 5 2 + - +

4

2. 5 7 + 2 * 8 -

16

3. 2 3 8 + * 2 4 - *

-44

4. 4 8 6 2 3 - + / *

6.4

5. 4 5 / 2 6 - 3 2 - - *

-4

Weekly assignments

- Set **every Wednesday morning** on Google Classroom
- Due the following **Tuesday at 16:30**
- As a minimum, you are expected each week to
 - Complete 1 hour of on-going revision using provided flashcards
 - This will not start until week 5
 - Until then, you will be expected to complete the programming worksheets instead
 - Answer theory questions
 - Complete practical programming work and submit code and testing screenshots
 - Self-mark your answers
 - Reflect and make notes on improvement

Assignment grades

Assignment GRADES 	
3	Completed to a good standard, marked and handed in on time.
2	Completed to a good standard and marked, but LATE .
1	INCOMPLETE and/or completed to a poor standard. RESUBMIT .
0	MISSING . Submit within 7 days.

Lunchtime workshops

Monday	Tuesday	Wednesday	Thursday	Friday
Tristan – N10	Alex – N10	Isabel – N10		

- Every Tuesday, Wednesday and Thursday lunchtime (starting next week)
 - 12:00–12:45
 - Drop in for as long as you like during this time
- Reasons to come to workshop include
 - To get help with a difficult topic from a recent lesson
 - To revise a topic from a while ago
 - To get help with a weekly assignment
 - To get advice about future plans (UCAS, university, apprenticeships, personal statements etc.)
 - To discuss something interesting related to computer science
 - For a quiet place to work on a PC at lunchtime

Converting between infix and postfix

Recap

- Notation
 - **Operators:** +, −, *, /
 - **Operands:** 3, 4, 5
- **Infix** is the notation you are most familiar with
 - Operators are placed between operands
 - e.g. $2 + 2$
 - Requires BIDMAS (or other rules) to evaluate
- **Postfix/reverse polish notation** is the notation we introduced last lesson
 - Operators are placed after operands
 - e.g. $2\ 2\ +$

Evaluating postfix

3 4 5 + -

Starting at the leftmost part of the expression,

1. Move right until you reach an **operator**
2. Apply the operator to the previous two **operands**
3. Repeat until the expression is fully evaluated

3 4 5 + -

3 9 -

-6

Advantages of postfix

Discuss: Why might postfix be better for computers than infix?

- There is no need for an order of precedence of operations (e.g. BIDMAS)
 - This is because postfix expressions can be evaluated left-to-right
- A computer can evaluate a postfix expression faster than the same expression in infix
- There is no need for brackets

Converting infix to postfix

$$6 + 7 * 3$$

1. Fully bracket the expression
2. Move operators to right of *their* bracket
3. Remove the brackets

$$6\ 7\ 3\ *\ +$$

Warning

Two expressions can have the same value without being the same expression.

Examples

1. Fully bracket the expression
2. Move operators to right of *their* bracket
3. Remove the brackets

1. $(2 + 5)/6$

2. $4 + (9/3 + 2)$

$2\ 5 + 6/$

$4\ 9\ 3/2 + +$

Tip: Operands maintain the same *relative* order

Practice



Convert
 $8 + 2 * 5$
into postfix notation

$8\ 2\ 5\ *\ +$



Practice

Convert
 $9 - (4 + 2) * 2$
into postfix notation

$9\ 4\ 2\ +\ 2\ *\ -$



Practice

Convert
 $7 - 6/2 + 5$
into postfix notation

$7\ 6\ 2/\ -\ 5\ +$
or $7\ 6\ 2/5\ +\ -$

Converting postfix to infix

$$3\ 4\ +\ 2\ *$$

Starting at the leftmost part of the expression,

1. Move right until you reach an **operator**
2. Move the operator between the previous two **operands** and surround with brackets
3. Repeat until the expression is fully evaluated

In this case, bracketed expressions count as operands!

$$(3 + 4) * 2$$

Using a stack

- A stack is a special way of storing data
- Much like a stack of books, you can place values on top of the stack and take them off, but you can't access the middle of the stack

3 4 + 2 *

Examples

Starting at the leftmost part of the expression,

1. Move right until you reach an **operator**
2. Move the operator between the previous two **operands** and surround with brackets
3. Repeat until the expression is fully evaluated

1. 6 2 7 / -

$6 - 2 / 7$

2. 8 3 5 + 2 / *

$8 * (3 + 5) / 2$

Tip: Bracketed expressions count as operands

Practice



Convert
 $8\ 2\ 3\ *\ -$
into infix notation

$8 - 2 * 3$

Practice



Convert
 $3\ 5\ +\ 2\ /$
into infix notation

$$(3 + 5) / 2$$



Practice

Convert
 $8\ 5\ 2\ +\ 2\ *\ -$
into infix notation

$$8 - (5 + 2) * 2$$

Exercise

1. $4 + 2 * 9$

$4\ 2\ 9\ *\ +$

2. $(8 - 5)/3$

$8\ 5\ -\ 3\ /$

3. $4 + 7 - 2 * 3$

$4\ 7\ +\ 2\ 3\ *\ -$

4. $(9 - 2)/((3 + 9)/4)$

$9\ 2\ -\ 3\ 9\ +\ 4\ /\ /$

1. $4\ 9\ 5\ +\ *$

$4\ * (9 + 5)$

2. $3\ 6\ 2\ 1\ +\ * -$

$3 - 6 * (2 + 1)$

3. $6\ 8\ +\ 2/3\ +$

$(6 + 8)/2 + 3$

4. $8\ 3\ 6\ +\ 3/\ +\ 9\ 4\ -\ * 5 -$

$(8 + (3 + 6)/3) * (9 - 4) - 5$

Exam question

0 5

$(3 + 4) * 5$ is an example of an infix expression. The same expression has been represented in a different expression format in **Figure 2**.

Figure 2

3 4 + 5 *

0 5 . 1

What is the name of the expression format used in **Figure 2**?

[1 mark]

0 5 . 2

Represent the infix expression $5 + 2 * 3 + 4$ in the same expression format used in **Figure 2**.

[2 marks]

Exam question (mark scheme)

Question			Marks
05	1	Mark is for AO1 (understanding) Reverse Polish (Notation) // RPN; A. Postfix	1
05	2	All marks AO2 (apply) 523*+4+;; Mark as follows: 1 mark: 23* in expression; 1 mark: correct order of operands with + symbols either side of the 4; Max 1 mark if any errors	2

Topic 4.1 – Programming

Selection

Recap

Discuss:

What is a variable?

What does every variable have?

Practice



Write the line of code required to create a variable called `name` that stores the name of the person sitting next to you.

```
string name = "Alex";
```

Practice



Write the line of code required to create a variable called `lessons` that stores the number of lessons you have today.

```
int lessons = 3;
```

Practice



Write the line of code required to create a variable called `bestSubject` that stores a value input by the user.

```
string bestSubject = Console.ReadLine();
```

Practice



Write the line of code required to create a variable called `mealDealPrice` that stores the value 3.5

```
double mealDealPrice = 3.5;
```

Practice



Write the line of code required to create a variable called `isWarmToday` that stores either true or false.

```
bool isWarmToday = true; // hopefully!
```

Selection & if statements

- **Selection** is the programming concept by which we choose a path in the code to take depending on a condition
- A **condition** is an expression that evaluates to true or false
 - e.g. `isWarmToday`
 - e.g. `age > 18`
- An **if statement** is a form of selection
 - A block of code runs only if a specified condition is true

Conditions

- A condition is an expression that evaluates to a boolean (true or false)
- Comparisons are a form of condition

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
!=	Not equal to
==	Equal to



Comparisons practice

Let $A = 1$, $B = -2$, $C = 3$, $D = 4$, $E = 'S'$ and $F = 'J'$.

State whether the following conditions are true or false.

- | | |
|--|--------------|
| 1. $A == B$ | False |
| 2. $A > B$ | True |
| 3. $(A < C)$ and $(B > D)$ | False |
| 4. $(A < C)$ and $(B < D)$ | True |
| 5. $(A < B)$ or $(C < D)$ | True |
| 6. $E > F$ | True |
| 7. $((A + C) > (B - D))$ and $((B + C) < (D - A))$ | True |

Demo

```
int num;

Console.Write("Enter number: ");
num = int.Parse(Console.ReadLine());

if (num > 0)
{
    Console.WriteLine(num + " is positive");
}
else
{
    Console.WriteLine(num + " is negative");
}
```

Logical operators

- We can combine conditions with **logical operators**
- && is used for logical AND
- || is used for logical OR

```
int x = 5;
```

```
if (x >= 1 && x <= 10)
{
    Console.WriteLine("x is in range!");
}
```

Selection

Worksheet

W101 - C# If