

Workbook



Table of Contents

Fundamental Programming Concepts.....	2
Introduction	2
Basic syntax and semantics of a HLL	3
Variables and primitive data types.....	4
Assignment And Expressions.....	6
Conditional Control Structures	9
Iterative Control Structures.....	15

Fundamental Programming Concepts

Introduction

Questions

- 1) When describing the CPU, why do Computer Scientists not like using the brain as an analogy and what makes a calculator a better analogy?
- 2) Where are instructions stored, and what other components are needed in addition to a CPU to make the system we call a "computer"?
- 3) How does an instruction set relate to the concept of binary numbers?
- 4) Explain two advantages and one disadvantage of writing code in assembly language rather than in machine language.
- 5) Computer systems may use CPUs from a variety of different manufacturers.
 - a. Why is code written in low level languages for differing CPUs incompatible?
 - b. How do high level languages solve this problem?
 - c. Why are there so many high level languages?
- 6) High level languages require translation using interpreters or compilers. Describe the relative advantages and disadvantages of each method.
- 7) What is the difference between a programming language and a programming paradigm?

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Basic syntax and semantics of a HLL

Questions

- 1) Languages can be described as natural or formal
 - a. Describe two advantages of natural languages over formal languages
 - b. What feature of natural languages makes formal languages necessary for programming?
- 2) Syntax can be defined as "the set of rules that define the precise structure of a restricted number of words and symbols in a language".
 - a. To what degree does the above definition apply for a natural language?
 - b. Describe, giving examples, how it is possible for an expression in a language, to use the correct syntax but be invalid.
 - c. What advantage do humans using natural languages have over machines, using formal languages in terms of longer sequences of expressions and statements?
- 3) A compiler or interpreter may report a syntax error
 - a. Why might "compile time error" be a better way of describing these types of error?
 - b. How is the term syntax error sometimes used to describe a different type of error, and why is this not strictly correct?
- 4) Syntax errors are not the only type of error
 - a. Why is it not possible to pick up run time errors in advance of the program being run?
 - b. Why is it that experienced programmers spend more time dealing with logical errors than syntax errors?
 - c. The amount of free memory available whilst a program is running can vary. What type of error could this cause and why?

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Variables and primitive data types

Questions

- 1) Create a new project, and open the Code Pad in BlueJ. Do the following activities in the Code Pad:
 - a. Create a new variable called `found` that only stores boolean values.
 - b. Store a boolean value in `found`.
 - c. Check the boolean value is stored correctly by typing `found` followed by the enter key.
 - d. Try to store an integer value in `found` and see what happens. Can you explain why this happens?

- 2) In the same Code Pad:
 - a. Create a new variable to store a 7 bit signed integer value using the byte data type, and a sensible name.
 - b. Store the value 10 decimal into the variable.
 - c. Check the value is stored correctly by typing the variable name followed by the enter key.
 - d. The prefix `0x` can be used to store a hexadecimal value into the variable rather than decimal. Store the hexadecimal value 5F into the variable.
 - e. Check that the value 95 decimal has been successfully stored into your variable.
 - f. Now try to put the hexadecimal value AF into the variable. Can you explain what happens?
 - g. The prefix `0b` can be used to store a binary value into a variable. Store the binary value `01011100` into your variable and confirm that the value decimal 92 has been stored.
 - h. Now store the binary value `10000000` into the variable. Can you explain what happens?

- 3) In the Code Pad again:
 - a. Create a variable in Code Pad called `myInt` to store an integer.
 - b. Store the maximum negative value possible (hint: Java integer datatypes are signed and use 32 bits).
 - c. Type `myInt` and press enter to check that your intended value was stored.
 - d. Now store the maximum positive value possible into `myInt`. Again, check that the intended value was stored.
 - e. With the maximum positive value stored, type the following, which will add one to the contents of the variable: `myInt +=1`.
 - f. The result should be a negative number. Can you explain why this happens?

Principles of Programming

4) Continuing in the Code Pad:

- a. Create a new variable called `myLong` to store a long integer.
- b. Store into `myLong` the number 2,147,483,648 (hint: integer literals in Java will be assumed to be for the `int` datatype, unless you append an `L` to the end of the literal).
- c. Check the number you expect is stored in `myLong`.
- d. Store the maximum positive number possible into `myLong`.
- e. If you were to add one to the value stored in part d above, what would you expect to happen?

5) Finally, and still in the Code Pad:

- a. Create four more different variables using the remaining primitives available in Java, that we haven't already used in the previous exercises.
- b. Give each variable a sensible name, but use a mixture of styles (e.g. camel case, snake case, capital camel case, and capital snake case).
- c. Store values into each, and see the effect.
- d. Try to cause an error by attempting to store an illegal value in each.

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Assignment And Expressions

Questions

- 1) Create a new project, and open the Code Pad in BlueJ. Do the following activities in the Code Pad:
 - a. With a single line of code create a boolean variable called `isValid` and initialise it with the value `true`.
 - b. Press the up arrow key to retrieve the line you just typed in.
Change the last part of the line so the value is `false`, not `true`.
Can you explain why you get an error message when you press enter to compile the line?
 - c. Correct the line to successfully change the value to `false`.

- 2) Within the same Code Pad:
 - a. Create two integer variables, `num1` and `num2` with no initial value given.
 - b. Assign the value `10` to `num1`.
 - c. Assign the contents of `num1` to `num2`.
 - d. Confirm the contents of `num2` are what you expect.

- 3) Now create another pair of variables in the Code Pad:
 - a. Create a char variable and assign it the value `'a'`.
 - b. Create a `short` variable with no initial value.
 - c. Assign the contents of the char variable to the `short` variable. Can you explain what happens?
 - d. Now try again but in *front* of the char variable name put the following characters exactly as they appear here: `(short)`
 - e. Look at the contents of your `short` variable. Can you explain why it contains that value?
 - f. Now assign the Unicode character `'\u9000'` to your char variable.
 - g. Once again, assign the value of your char variable to the `short` variable, using `(short)` in front of the char variable name.
 - h. Look at the contents of your `short` variable. Can you explain why it contains that value?
What has gone wrong?

- 4) Create a third integer variable in Code Pad called `num3` to complement the two you already created in exercise 2 above.
 - a. assign the value `10` to `num1`, `20` to `num2` and `4` to `num3`. Check the contents of all three variables.
 - b. Create a single expression involving all three variables that adds `num1` to `num2` divided by `num3`. Note the result of the evaluation.
 - c. Put brackets around the addition of `num1` and `num2` and explain the resulting evaluation.
 - d. Using the same expression as part c, use the modulo operation instead of the divide operation, and explain the resulting evaluation.

- 5) Using the same variables as exercise 4 above, do the following in the Code Pad:
- Assign the value 7 to variable num1 and 8 to variable num2
 - Without using any other variables, use the comparison/relational operators in the Code Pad to test if:
 - num1 and num2 are equal
 - num1 and num2 are not equal
 - num1 is greater than num2
 - num1 is less than num2
 - Change the value of num2 to 7 and then test if:
 - num1 is greater than or equal to num2
 - num1 is less than or equal to num2
 - Assign the value 9 to num3.
 - Use the logical and operator to create a single expression that evaluates whether num2 is greater than 5 and less than 10.
 - Change the value of num2 to 5, and run the expression you created in part e again.
 - Use the logical or operator to create a single expression that evaluates whether num1 is less than num2 or num3 is equal to 11.
 - Change the value of num3 to 11, and run the expression from part f again to see if the evaluation changes.
 - Create your own expression using a combination of logical and comparison operators. Include the not operator and brackets to change the order of evaluation. Change the value of at least one of your variables and see the effect on the evaluation.
- 6) In the Code Pad as before:
- Create a variable using the byte datatype and assign it the value 65.
 - Perform a bitwise and operation on your variable using the binary value 0b01000000.
 - Can you explain why you get the resulting value?
 - Perform a bitwise or operation on your variable using the decimal value 32.
 - Can you explain why you get this new value?
 - Assign the value 17 to your byte variable, then perform a bitwise exclusive or operation on your variable using the binary value 0b00010001.
 - Can you explain the resulting evaluation?
 - Assign the value 1 to your byte variable, then perform a complement operation on the variable. Can you explain the value you get? (hint: a byte stores a signed two's complement 7 bit value).

Principles of Programming

- 7) Create a variable of `int` datatype, and assign it the initial value `10` decimal.
- Use the post increment operator on the variable and don't put a semi colon on the end of the line.
 - Why does the value reflected back by BlueJ not appear to change?
 - Use the pre increment operator on the variable and once again, don't put a semi colon on the end of the line
 - What difference do you observe between post and pre increment?
 - Use the pre decrement operator on your variable.
 - Use the post decrement operator on your variable.
 - Use a compound assignment to multiply the variable by `10` and store the result back into the variable.
 - Use a compound assignment to add `28` to the variable and store the result back into the variable.
 - Use a compound assignment to do a bitwise exclusive or with your variable and the value `129` and store the value back into your variable.
- 8) The final exercise should be completed as a BlueJ compiled program, so create a class called `MyTempConverterApp` inside your project. Inside the class:
- Declare a variable of an appropriate type to hold an input temperature, and give it the initial value `37.77778`.
 - Declare a variable of an appropriate type to hold an output temperature.
 - Create a method inside your class called `convert()`.
 - Within the `convert()` method, add a line that uses whatever operators are needed to convert the input temperature from Celsius to Fahrenheit, and to assign the evaluated value to the output temperature variable.
 - The second line in the method should output the output temperature to the screen using `System.out.println`.
 - Compile your program and correct any errors signalled by the compiler.
 - Test your program by creating an object from your class.
 - Run the `convert()` method and check your result is `100.0`.

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Conditional Control Structures

Questions

- 1) Create a new project in BlueJ, and a new class inside it called ConditionalsApp.
 - a. Create a new method called ageWarning().
 - b. Create a variable called age and initialise it with the value 21.
 - c. Use a conditional control structure to print the message "Renters under the age of 25 must provide an additional security deposit via credit card" but only when the age variable contains a value less than 25.
 - d. The method should always output a message saying "Minimum rental age is 21".
 - e. Compile your program, correcting any errors, and test by creating an object from the ConditionalsApp class on the object workbench and running the ageWarning() method.
 - f. Change the value of the age variable to 25 and run the ageWarning() method again.

- 2) In the same class:
 - a. Create a new method called welcome()
 - b. Within welcome() create two variables called age and hasAccount initialised with the values 25 and true.
 - c. Use a conditional control structure to print the output "You are eligible to use the fast collect service." only when age is 25 or over, and hasAccount is true.
 - d. The method should always output a message saying "Thank you for choosing Hello World Rent-a-car." after any previous message is printed.
 - e. Compile your program, correcting any errors, and test by creating an object from the ConditionalsApp class on the object workbench and running the welcome() method.
 - f. Change the values of the age and hasAccount variables to values that will more thoroughly test the program.

- 3) Create a new method called `rentalCost()` which will display a message that calculates the cost of a rental given a number of days and a daily rate.
- a. Declare and initialise:
 - i. a variable called `rate` with the value 12.
 - ii. a variable called `hasAccount` with the value `true`.
 - iii. a variable called `rentalLength` with the value 3.
 - b. Create a single conditional structure that outputs different messages depending on the following conditions:
 - i. If `hasAccount` is not `true`, output a message saying "Your rental cost:" followed by the output from an expression which calculates the total rental figure by multiplying the rental length and the daily rate.
 - ii. If `hasAccount` is `true`, output a message saying "Member's 25% discount applied. Your rental cost:" followed by the output from an expression which will give a total rental figure with a 25% discount applied.
 - c. Test the `rentalCost()` method to see if it outputs the value 27, then change the variable `hasAccount` to `false` and test again to see if you get the value 36.

Principles of Programming

- 4) Create a new method called `multiDiscount()` which again takes into account the daily rate and rental length to calculate the total rental cost, but this time applies a discount to members, and a further discount to people over 60.
 - a. Declare and initialise
 - i. a variable called `age` with the value 67
 - ii. a variable called `rate` with the value 12
 - iii. a variable called `hasAccount` with the value `false`
 - iv. a variable called `rentalLength` with the value 5
 - b. Use multiple conditional statements to do the following:
 - i. For renters aged over 20 but under 25, display a message saying "Renters under the age of 25 must provide an additional security deposit via credit card."
 - ii. For renters over age 20 but under 60 who have an account, display a message saying "Member's 25% discount applied. Total rental cost:" followed by an expression that will display the rental cost with a 25% discount applied
 - iii. For renters over age 20 but under 60 who do not have an account, display a message saying "Total rental cost:" followed by an expression that will display the rental cost with no discount applied
 - iv. For renters of 60 or over, with no account, display a message saying "Seniors 10% discount applied. Total rental cost:" followed by an expression that will display the rental cost with a 10% discount applied
 - v. For renters of 60 or over, who have an account, display a message saying "* Senior member's multi discount applied * Total rental cost:" followed by an expression that will display the rental cost with a 30% discount applied
 - c. Test the `multiDiscount()` method to confirm whether it outputs a message saying: Seniors 10% discount applied. Total rental cost: 54
 - d. Change the `hasAccount` variable to contain `true` and again test the `multiDiscount()` method to confirm whether it outputs a message to say:
* Senior member's multi discount applied * Total rental cost: 42
 - e. Change the `age` variable to contain 21 and once again test the `multiDiscount()` method to confirm whether it outputs a message to say :
Renters under the age of 25 must provide an additional security deposit via credit card. Member's 25% discount applied.
Total rental cost: 45
 - f. Change the `hasAccount` variable back to `false` and one final time test the `multiDiscount()` method to confirm whether it outputs a message to say:
Renters under the age of 25 must provide an additional security deposit via credit card. Total rental cost: 60

Principles of Programming

- 5) Create a method called `multiDiscount2()` to perform the same task as the previous exercise but using nested conditional structures instead of linked conditional statements
- a. Declare and initialise
 - i. a variable called `age` with the value 18
 - ii. a variable called `rate` with the value 12
 - iii. a variable called `hasAccount` with the value `false`
 - iv. a variable called `rentalLength` with the value 5
 - b. Check whether the renter is between 21-24:
 - i. If they are, then display the message: "Renters under the age of 25 must provide an additional security deposit via credit card." and output the total cost with discount applied dependant on whether they have an account or not.
 - ii. If they are not between 21-24, perform a further nested check to see whether they are over 59, or between 25 and 59, and apply the correct discounts taking into account whether they have an account or not.
 - c. Test `multiDiscount2()` which should display nothing if everything has been implemented correctly because `age` contains 18. Correct your code if this is not the case.
 - d. Change `age` to contain 21 and once again test the `multiDiscount2()` method to confirm whether it outputs a message to say:
`Renters under the age of 25 must provide an additional security deposit via credit card. Total rental cost: 60`
 - e. Change `age` to contain 65 and once again test the `multiDiscount2()` method to confirm whether it outputs a message to say:
`Seniors 10% discount applied. Total rental cost: 54`
 - f. Change the `hasAccount` variable to contain `true` and again test the `multiDiscount2()` method to confirm whether it outputs a message to say:
`* Senior member's multi discount applied * Total rental cost: 42`

6) Study the code below:

```
char prizeCode='b';

switch (prizeCode){
    case 'a' :
        System.out.println("Teddy bear");
        break;
    case 'b' :
        System.out.println("Headphones");
        break;
    case 'c' :
        System.out.println("Game");
        break;
    default :
        System.out.println("Invalid prize");
        break;
}
```

One of the lines above can be removed without affecting the operation of the code. Which one is it, and why does removing it make no difference?

7) Create a new method called `multiDiscount3()`

- a. Declare and initialise
 - i. a variable called `rate` with the value 12
 - ii. a variable called `rentalLength` with the value 10
 - iii. a variable called `discountCode` with the value 0
- b. Create a switch structure which outputs the total rental cost and applies different discounts depending on what value is in `discountCode` as follows
 - i. 0 gives no discount so is the standard price
 - ii. 1 gives a senior discount of 10%
 - iii. 3 gives a member discount of 25%
 - iv. 5 gives a multi discount of 30%
- c. If `discountCode` contains none of the above values, display a message saying "Invalid discount code applied. Please try again."
- d. Test your code with values of
 - i. 0 – which should give the result 120
 - ii. 1 – which should give the result 108
 - iii. 3 – which should give the result 90
 - iv. 5 - which should give the result 84
 - v. 2 – which should display the invalid discount code message

- 8) Edit the method `multiDiscount3()` from the previous exercise:
- Add in a variable called `age` and initialise it with the value `21`.
 - Change the initial value for `discountCode` to `0`.
 - Add a conditional statement into the switch structure which displays a warning message for `discountCode 0` only, where for ages between 21 and 24, users are shown the message "Renters under the age of 25 must provide an additional security deposit via credit card."
 - Test the code works as expected.

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Iterative Control Structures

Questions

- 1) Create a new project in BlueJ, and a new class inside it called `IterationApp`
 - a. Create a new method called `forCountUp()`.
 - b. Using a for loop, output the numbers from 1 to 10.
 - c. Create another method called `forCountDown()`.
 - d. Using a for loop, output the numbers from 5 to 1, ending on 1.
 - e. Test both methods by creating an object on the workbench, and correct any errors found.

- 2) In the same class
 - a. Create a new method called `forCountIn2s()`.
 - b. Using a for loop, output the numbers from 2 to 24 in steps of 2.
 - c. Create a new method called `forCountIn10s()`.
 - d. Using a for loop, output the numbers from 100 to 0 in steps of -10.
 - e. Test both methods by creating an object on the workbench, and correct any errors found.

- 3) Create a new method called `sevenTimesTable()`:
 - a. Create an integer variable called `table` and initialise it with the value 7, and a second integer variable called `multiplier` with no initial value.
 - b. Use a for loop to iterate the `multiplier` variable through the numbers 1 to 12.
 - c. Inside the for loop, output the contents of the `multiplier` variable followed by the string " x " followed by the contents of the `table` variable, followed by the string " = ".
 - d. On the same line as the previous output, output the product of the `multiplier` and `table` variables.
 - e. Test your method by creating an object on the workbench, checking that all twelve lines of the seven times table are correctly output. Correct any errors found.

- 4) Create a new method called `allTimesTables()`
- Create an integer variable called `table` with no initial value, and a second integer variable called `multiplier` with no initial value.
 - Create a pair of nested for loops. Both of which iterate between 1 and 12.
The inner loop should iterate `multiplier`, and the outer loop should iterate `table`.
 - At the appropriate point in the code, output the contents of the `multiplier` variable followed by the string " x " followed by the contents of the `table` variable, followed by the string " = ".
 - On the same line as the previous output, output the product of the `multiplier` and `table` variables.
 - Check that the unlimited buffering option is ticked in the terminal window of BlueJ, then test your method by creating an object on the workbench, checking that all twelve lines of the 1 times table to the 12 times table are correctly output (144 lines in total). Correct any errors found.
 - At an appropriate point in the code, output the contents of the `table` variable followed by the string " times table:" so that each table is preceded by a heading indicating which times table is being output.
- 5) Create a new method called `whileCount()`:
- Using a while loop, output the numbers from 1 to 10
 - Test your method. If it continues producing output past 10 or appears to have crashed, click on the reset virtual machine icon in the bottom right corner of the BlueJ application window. Correct any errors found.
 - In the same method, add another while loop after the previous one which outputs the numbers from 10 to 100 in steps of 10.
 - Test your method again. If it continues producing output past 100 or appears to have crashed, click on the reset virtual machine icon in the bottom right corner of the BlueJ application window. Correct any errors found.
 - In the same method, add another while loop after the previous one which outputs the numbers from 2020 to -5000 in steps of -100.
 - Test your method again. If it continues producing output past -5000 or appears to have crashed, click on the reset virtual machine icon in the bottom right corner of the BlueJ application window. Correct any errors found.

Principles of Programming

- 6) Create a new method called `fibonacci()`
- Declare and initialise
 - a variable called `num1` with the value 0
 - a variable called `num2` with the value 1
 - a variable called `limit` with the value 12
 - a variable called `temp` with the initial value 0
 - a variable called `term` with the initial value 0
 - Use a while loop to iterate while `term` is less than or equal to `limit`.
 - Inside the loop:
 - output `num1`
 - assign to `temp` the sum of `num1` and `num2`
 - assign to `num1` the contents of `num2`
 - assign to `num2` the contents of `temp`
 - increment the `term` variable
 - Test your method outputs the Fibonacci sequence:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
- 7) Create a new method called `whileTimesTables()`
- Create an integer variable called `table` with an initial value of 2, and a second integer variable called `multiplier` with an initial value of 1.
 - Create a pair of nested while loops. The inner loop should iterate `multiplier` from 1 to 12, and the outer loop should iterate `table` from 2 to 12.
 - At the appropriate point in the code, output the contents of the `multiplier` variable followed by the string " x " followed by the contents of the `table` variable, followed by the string " = ".
 - On the same line as the previous output, output the product of the `multiplier` and `table` variables.
 - Check that the unlimited buffering option is ticked in the terminal window of BlueJ, then test your method by creating an object on the workbench, checking that all twelve lines of each table are correctly output. Correct any errors found.
 - Add an integer variable called `line`, and display its contents followed by the literal string " : " before each line of the times tables is output to the screen.
 - Test the method, checking that all 132 line numbers are printed next to each line. Correct any errors found.

- 8) Create a new method called `doWhileCount()`
- a. Using a `do...while` loop, output the numbers from 1 to 10 .
 - b. Test your method. If it continues producing output past 10 or appears to have crashed, click on the reset virtual machine icon in the bottom right corner of the BlueJ application window. Correct any errors found.
 - c. In the same method, add another `do...while` loop after the previous one which outputs the numbers from 10 to 100 in steps of 10.
 - d. Test your method again. If it continues producing output past 100 or appears to have crashed, click on the reset virtual machine icon in the bottom right corner of the BlueJ application window. Correct any errors found.
 - e. In the same method, add another `do...while` loop after the previous one which outputs the numbers from 2000 to -5000 in steps of -100.
 - f. Test your method again. If it continues producing output past -5000 or appears to have crashed, click on the reset virtual machine icon in the bottom right corner of the BlueJ application window. Correct any errors found.

- 9) Create a new method called `primeNumbers()`
- Declare and initialise:
 - a variable called `number` with the value 2.
 - a variable called `factorsFound` with the value 0.
 - a variable called `limit` with the value 20.
 - a variable called `factor` with no initial value.
 - Create a `do...while` loop, which terminates when `number` is no longer less than or equal to `limit`.
 - Within the `do...while` loop above:
 - assign the value of `number` to `factor`.
 - create another `do...while` loop which terminates when `factor` is no longer greater than or equal to 2.
 - Within the inner `do...while` loop determine whether `number` divided by `factor` gives a remainder of 0, and increment the `factorsFound` variable when it does.
 - Within the inner `do...while` loop but after the conditional statement in part iii above, decrement the `factor` variable.
 - Directly after the inner `do...while` loop, but inside the outer `do...while` loop:
 - Check if `factorsFound` is equal to 1, and if so output a message showing the current value of the `number` variable followed by the literal string " is a prime".
 - increment the `number` variable.
 - Assign the value 0 to `factorsFound`.
 - Test the method produces an output of all the prime numbers below 20: 2, 3, 5, 7, 11, 13, 17, 19.
Correct any errors found.
 - Change the initial value of `limit` to 100. Test the method produces an output of all the prime numbers below 100:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.
Correct any errors found

Answer Key

To view the answers to these exercises please refer to the appropriate videos.