1 Fundamentals of programming

1.1 Programming

Updated 11th February 2019

Chapter 1.1.1 Data types

|  |  |
| --- | --- |
| 1a) | String |
| 1b) | Integer |
| 1c) | Character |
| 2) | The string “Hi”; the integer 18537 |
| 3) | No |
| 4) | A data type is a correspondence between a mathematical entity such as integer and a set of datums. |
| 5) | No |
| 6a) | 62,471,600,000 |
| 6b) | 86,114,100,000 |
| 6c) | 23,642,400,000 |
| 6d) | 62,471,600,000 |
| 6) | The two answers would differ as the results of x2 and y2 can’t be accurately represented using 9 or fewer significant digits. |
| 7a) | Either could be used as 45 can be represented precisely using both of these data types. |
| 7b) | 32-bit integer would be used as IEEE 754 can’t use more than 9 significant digits. |
| 7c) | IEEE 754 would be used as the number is too large to represent as a 32-bit integer. |
| 8a) | IEEE 754 as it is not an integer. |
| 8b) | IEEE 754 as number is not an integer (and it is too close to 0 to be represented accurately as a 32-bit integer). |
| 8c) | IEEE 754 as number is too large to be represented as a 32-bit integer. |
| 9a) | Boolean |
| 9b) | Boolean |
| 9c) | Boolean |
| 10a) | Y |
| 10b) | H |
| 10c) | L |
| 10d) | y |
| 11) | All three could be successfully evaluated as they are comparing two values of same data types. |
| 12) | Because it allows a group of characters to be stored in an order. |
| 13) | Date |
| 14) | A pointer is a memory address. |
| 15) | The memory location allocated to the pointer stores a memory address that contains the integer value. |
| 16) | Record  Title : String  ISBN: String  InStock: Boolean  NoOfPages: Integer  Price: Real  Category: Character  End |
| 17) | An array data type is a (fixed-size if static) sequential collection of elements of the same type. |
| 18) | TArrayOfReals = Array[0..4] Of Real |
| 19) | A user-defined data type is one built by a programmer. |

Chapter 1.1.2 Programming concepts

|  |  |
| --- | --- |
| 1) | A container in a program which can be used to store a value.  A named given to one or more memory locations used to contain a value while a program is running. |
| 2) | A variable declaration is a way of creating a variable. |
| 3) | Constant declaration is giving a symbolic name to a data item which will never change. |
| 4) | It makes the program code easier to understand.  It means there is no chance of inconsistency i.e. typing in different values for the constant in different parts of the program. |
| 5) | Assignment is an instruction that alters the value of a variable. |
| 6) | |  |  | | --- | --- | | NatNo | RunningTotal | | 1 | 0 | | 2 | 1 | | 3 | 3 | | 4 | 6 | | 5 | 10 | | 6 | 15 | | 7 | 21 | | 8 | 28 | | 9 | 36 | | 10 | 45 | | 11 | 55 | |
| 7) | Change Until NatNo =11 to Until NatNo = 16. |
| 8) | |  |  | | --- | --- | | NatNo | RunningTotal | | 1 | 0 | | 2 | 1 | | 3 | 3 | | 4 | 6 | | 5 | 10 | | 6 | 15 | | 7 | 21 | | 8 | 28 | | 9 | 36 | | 10 | 45 | | 11 | 55 | |
| 9) | Change While NatNo < 11 to While NatNo < 16 (or to While NatNo <= 15). |
| 10) | 3,628,800   |  |  | | --- | --- | | Count | RunningTotal | | 2 | 1 | | 3 | 2 | | 4 | 6 | | 5 | 24 | | 6 | 120 | | 7 | 720 | | 8 | 5,040 | | 9 | 40,320 | | 10 | 362,880 | | 11 | 3,628,800 | |
| 11) | Temp 🡨 x  x 🡨 y  y 🡨 Temp  Alternatively,  Temp 🡨 y  y 🡨 x  x 🡨 Temp |
| 12) | If Age < 37 Then  AgeCategory 🡨 'A'  Else  AgeCategory 🡨 'B'  Endif  Alternatively,  If Age >= 37 Then  AgeCategory 🡨 'B'  Else  AgeCategory 🡨 'A'  Endif |
| 13a) | The Then part because No1 is larger than No2. |
| 13b) | The Else part because No1 is not larger than No2. |
| 14) | A procedure interface is a mechanism for passing data into and out of a procedure. |
| 15) | A function always returns a single result; a procedure can return no result or any number of results. |
| 16) | 2  4  6  8  10 |
| 17) | A  B  C |
| 18) | One |
| 19a) | 10 |
| 19b) | 24 |
| 20a) | 3  2  1  0 |
| 20b) | 5  6  7 |
| 20c) | 5 |
| 21) | With definite iteration the number of times to repeat is known before the loop starts; with indefinite iteration the number of times to repeat is not known – it keeps going until a condition is met. |
| 22) | A repeat until loop will iterate 1 or more times; a while loop will iterate 0 or more times. |
| 23) | An infinite loopis when the terminating (stopping) condition for a loop can never be met. |
| 24) | A 🡨 0  While A < 1  Output A  Endwhile |
| 25a) | Black, blue (then part). No1 is not bigger than No2 so it goes into the else part of the black if; No2 is bigger than No3 so it goes into the then part of the blue if. |
| 25b) | Black, blue (then part). No1 is not bigger than No2 (it is the same) so it goes into the else part of the black if; No2 is bigger than No3 so it goes into the then part of the blue if. |
| 26a) | 1  2  3  1  2  3 |
| 26b) | 1  1  2  1  2  3 |
| 27a) | A  A  B  A  B  C |
| 27b) | aaa  aab  aac  aba  abb  abc  aca  acb  acc  baa  bab  bac  bba  bbb  bbc  bca  bcb  bcc  caa  cab  cac  cba  cbb  cbc  cca  ccb  ccc |
| 28) | For i 🡨 1 To 12  Output '\*'  Endfor |
| 29) | |  |  |  | | --- | --- | --- | | i | j | Output | | 0 | 0 | 0 | |  | 1 | 1 | |  | 2 | 2 | | 1 | 0 | 0 | |  | 1 | 1 | |  | 2 | 2 | |
| 30) | Meaningful identifier names make program code easier for programmers to read and understand, they make the code self-documenting. |

Chapter 1.1.3 Arithmetic operations in a programming language

|  |  |
| --- | --- |
| 1a) | b^2-4\*a\*c |
| 1b) | 1/(1+x^2) |
| 1c) | 1/u+1/v |
| 2) | (a) 127 (b) 4 |
| 3) | 76 hours 29 minutes |
| 4) | N DIV 100 will give the number of 100s.  (N Mod 100) DIV 10 will give the number of 10s  (N Mod 100) MOD 10 will give the number of units |
| 5) | |  |  |  |  | | --- | --- | --- | --- | | **Dividend x** | **Divisor y** | **Quotient q** | **Remainder r** | | 5 | 2 | 2 | 1 | | 6 | 3 | 2 | 0 | | 25 | 4 | 6 | 1 | | 36 | 6 | 6 | 0 | | 121 | 7 | 17 | 2 | | 23 | 3 | 7 | 2 | | 1 | 3 | 0 | 3 | | 5 | 10 | 0 | 10 | |
| 6) | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Iteration** | **x** | **y** | **r** | **q** | **r > = y** | | 0 | 7 | 2 | 7 | 0 | True | | 1 |  |  | 5 | 1 | True | | 2 |  |  | 3 | 2 | True | | 3 |  |  | 1 | 3 | False | |
| 7) | Call the number X  Math.Truncate(X) will get the first digit, call this A  Math.Truncate(X\*10) – (A\*10) will get the second digit, call this B  Math.Truncate(X\*100) – (A\*100) – (B\*10) will get the third digit, call this C |
| 8) | Call the number X  Multiply X by 100.  Round the result to the nearest integer using Math.round.  Divide the result by 100. |

Chapter 1.1.4 Relational operations in a programming language

|  |  |
| --- | --- |
| 1a) | False |
| 1b) | False |
| 1c) | True |
| 1d) | False |
| 1e) | False |
| 1f) | True |
| 1g) | False |
| 2) | False |
| 3) | Have a nice evening! |

Chapter 1.1.5 Boolean operations in a programming language

|  |  |
| --- | --- |
| 1a) | False |
| 1b) | False |
| 1c) | True |
| 1d) | True |
| 1e) | False |
| 1f) | True |
| 1g) | False |
| 2a) | False |
| 2b) | False |
| 2c) | True |
| 2d) | False |
| 2e) | False |
| 2f) | True |
| 2g) | True |
| 3) | Have a nice evening! |

Chapter 1.1.6 Constants and variables in a programming language

|  |  |
| --- | --- |
| 1) | The contents of a variable can change while a program is running; after being given its initial value a constant does not change. |
| 2) | Named constants make it easier to avoid mistakes in programs arising from using different literal values in different places.  Named constants make programs easier to read.  Named constants make programs easier to modify/update. |

Chapter 1.1.7 String-handling operations in a programming language

No questions only programming tasks

Chapter 1.1.8 Random number generation in a programming language

|  |  |
| --- | --- |
| 1) | 1, 7, 10, 5, 9, 11, 12 |
| 2) | 12 |
| 3) | Because there are a limited number of values that can be returned by taking the modulus of a number (1 less than the number being used with the modulus operator) eventually the calculation will provide the same result. |
| 4) | To give a starting value for the random number generator. |
| 5) | If they know the programming language being used they will be able to find out the algorithm used to generate the pseudorandom numbers. They could have found out the starting seed value (using the time the game started on the server) – when this is known it is possible to recreate the sequence of pseudorandom numbers. |

Chapter 1.1.9 Exception handling

|  |  |
| --- | --- |
| 1) | An exception is an unexpected condition that arises during program execution. Exception handling moves error-handling code out-of-line; when an exception occurs control is transferred to the error handler which will either perform operation(s) to recover from the error or display an informative error message if this is not possible. |

Chapter 1.1.10 Subroutines(procedures/functions)

|  |  |
| --- | --- |
| 1) | The flow of control for a program using a subroutine is in line (top to bottom) until it encounters a subroutine call when control is passed to the subroutine (it is out of line). When the subroutine is finished control passes back to the line immediately after the subroutine call. To transfer control to a subroutine (to call it) you simply write the name of the subroutine in the appropriate place in the program code. |
| 2) | Subroutines can be reused (both within a program and in different programs.  Using subroutines makes a program easier for the programmer to understand and debug.  Different subroutines can be written by different programmers if a team of programmers are developing the program. |

Chapter 1.1.11 Parameters of subroutines

|  |  |
| --- | --- |
| 1) | A subroutine parameter is a used as a mechanism for passing data into (and, potentially, out of) a subroutine. |
| 2a) | The formal parameter gets a copy of the datum associated with the actual parameter used in the call to the subroutine. This means that the original version of the datum is unchanged by anything the subroutine does with the copy of the datum that it is given. |
| 2b) | The formal parameter is assigned the address in memory of the datum associated with the actual parameter. This means that the original version of the datum can be changed by the subroutine. |

Chapter 1.1.12 Returning a value/values from a subroutine

No questions

Chapter 1.1.13 Local variables in subroutines

|  |  |
| --- | --- |
| 1) | A local variable is a variable declared inside a subroutine and used within the body of the subroutine. |
| 2) | The lifetime of a local variable is the lifetime of the execution of the subroutine in which the variable has been declared. The scope of a local variable is the scope of the subroutine in which it has been declared – this is where it can be seen and accessed. |
| 3) | Using local variables is considered good practice as it aids modularization which allows each subroutine to be developed independently of other subroutines. Modularisation also allows subroutines to be reused in other programs.  Using local variables also prevents unintended side-effects that can arise from the use of global variables. |

Chapter 1.1.14 Global variables in a programming language

1) Local variables have a more limited scope than global variables – they can only be seen and accessed in part of the program.

Local variables are in memory only when the subroutine they are declared in is being executed, global variables are in memory the entire time the program is executing.

2)

That there is a local variable with the same identifier and the subroutine containing that local variable is currently being executed.

3)

Output will be 3 followed by 3. The reason for this is that when s was assigned the value 6 that was a local variable in the DoExample subroutine. The global variable s was not changed so the second output will also be 3.

Chapter 1.1.15 Role of stack frames in subroutine calls

See Chapter 2.1.4a

Chapter 1.1.16 Recursive techniques

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1) | Function Product(n: Integer) : Integer  If n = 1 Then  Return 1  Else  Return n + Sum(n – 1)  End If  End Function   |  |  |  | | --- | --- | --- | | **Call No** | **Product(n)** | **Return** | | 1 | Product(5) | 5 + Product(4) | | 2 | Product(4) | 4 + Product(3) | | 3 | Product(3) | 3 + Product(2) | | 4 | Product(2) | 2 + Product(1) | | 5 | Product(1) | 1 | |
| 2) | Function CountNoOfBallsInPyramid(n : Integer) : Integer  If n = 1 Then  Return 1  Else  Return NoOfBilliardBallsInTriangle(n) + CountNoOfBallsInPyramid(n – 1)  End If  End Function |
| 3ai) | Yes |
| 3aii) | 2 is a digit and 34 is a digit string because 3 is a digit and 4 is a digit string as 4 is a digit (base case). |
| 3bi) | No |
| 3bii) | 2 is a digit but .34 is not a digit string as it does not start with a digit (. is not a digit). |
| 4) | |  |  |  | | --- | --- | --- | | **Call no** | **Power(y, n)** | **Return** | | 1 | Power(2, 4) | 2 \* Power(2, 3) | | 2 | Power(2, 3) | 2 \* Power(2, 2) | | 3 | Power(2, 2) | 2 \* Power(2, 1) | | 4 | Power(2, 1) | 2 \* Power(2, 0) | | 5 | Power(2, 0) | 1 | |
| 5) | Because there is a limit on how much memory is allocated for the stack. Each time a function recurses another stack frame is added to the stack. Eventually, adding a stack frame could cause stack overflow. |
| 6) | |  |  |  | | --- | --- | --- | | **Call no** | **Power(y, n)** | **Return** | | 1 | Factorial(4) | 4 \* Factorial(3) | | 2 | Factorial (3) | 3 \* Factorial (2) | | 3 | Factorial (2) | 2 \* Factorial (1) | | 4 | Factorial (1) | 1 | |
| 7) | Function Factorial(n : NonNegativeInteger) : NonNegativeInteger  If n = 0 Then  Return 1  Else  Answer 🡨 1  For Count 🡨 1 To n  Answer 🡨 Answer \* Count  End For  End If  End Function |
| 8) | One |

1.2 Programming paradigms

Chapter 1.2.1 Programming paradigms

See Chapters 1.2.2 and 1.2.3

Chapter 1.2.2 procedural-oriented programming

|  |  |
| --- | --- |
| 1) | Design means to plan the form and method of solution. |
| 2) | Structured design is a process of deciding which components interconnected in which way will solve a problem. |
| 3) | Cohesion measures the strength of the interconnections between elements within modules. |
| 4a) | Because the procedure is not completing a single task (it has a compound sentence). |
| 4b) | Spilt the procedure into two – one procedure to add and one procedure to display. |
| 5) | The module can be replaced by another module which serves the same purpose.  Different developers can work independently on different modules.  It makes it easier to locate errors. |
| 6) | Coupling measures the strength of relationships between modules. |
| 7) | Minimising coupling ensures that modules are as independent as possible. |
| 8) | Coupling arises from connections between modules – when a code element accesses a memory location defined outside of its module. |
| 9) | To minimise coupling:   * Subroutines or modules should only be allowed to access that data which they needs to perform their assigned task * All data transfer between modules is visible in the module parameters * There must be no hidden flows of data via global variables or shared data areas * There should be no control information passing between modules, e.g. Boolean flags * The number of module parameters should be minimal. |
| 10) | There is a flow of data via global variables.  There is data transfer between modules that is not visible in the module parameters. |
| 11) | There is control information passing between modules. |
| 12) | Have two different subroutines (one which converts to upper case, one which doesn’t) |
| 13) | It uses too many parameters. |
| 14) | Low. Module B needs to know nothing about Module A in order to access file Z. It can read file Z independently of Module A. It may need to know the file structure (and file organisation) if it wishes to access the contents of file Z in a relatively “painless” way but it can always read file Z as a byte file, byte by byte. The file structure of file A (and organisation can be defined independently of both Module A and Module B. |
| 15) | Structured programming is a disciplined approach to the construction of programs. |
| 16) | The main flow of the program should be from top to bottom of the program. Every program block (sequence, selection, iteration) should have one entry point and one exit point. |
| 17) | Generally, it makes program code easier to understand and maintain. |
| 18) |  |
| 19) | It leads to functional cohesion which enables each module to be replaced by another serving the same purpose.  It leads to low coupling between modules making it easier to debug a module as it can be considered in isolation.  It makes it easier to understand the program code and check it meets its specification. |

Chapter 1.2.3(1) Object-oriented programming

|  |  |
| --- | --- |
| 1a) | An object is a collection of data plus the operations that act upon that data. |
| 1b) | Object instantiation is the name given to the process of creating an object. |
| 1c) | A constructor is a method used to grab the memory needed to create an object, they are also used to set initial values for an object’s attributes. |
| 1d) | An object is an instance of a class. |
| 2) | Objects share method code but not attribute data, each object maintains its own attribute values but there is only ever one copy of method code. An example would be if there are two house objects there would be two number of bedroom attributes (one for each object) but only one GetNumberOfBedrooms method. |
| 3) | It has been invoked. |
| 4) | Type  TQueueType = Class  Private  QueueArray : Array[1..100] Of Integer;  FrontPtr : Integer;  RearPtr : Integer;  CurrentSize : Integer;  Public  Function IsEmpty : Boolean;  Function IsFull : Boolean;  Function Delete : Integer;  Procedure Add;  Procedure ShowQueue;  Constructor Create;  Procedure ShowFrontPtr;  Procedure ShowRearPtr;  Procedure ShowCurrentSize;  End; |
| 5) | An object is an instance of a class and so can only exist when a program is actually being run. A class is a description which is used to create an object, it is part of the program source code. |
| 6a) | Implementation encapsulation is the hiding of the implementation of an object from the rest of the application. |
| 6b) | Data encapsulation means restricting access to the data (attributes) of an object. |
| 7ai) | A class is a template/blueprint from which objects are created. The class will define what attributes there are and what methods there are that act upon those attributes. |
| 7aii) | Inheritance allows a class to be derived from an existing class without the need to modify the existing class. The derived class has all the attributes and methods of the parent class, but adds new ones of its own. |
| 7b) | Member = Class  Private  MembershipNo : Integer  Name : String  Address: String  Public  Procedure ShowAddress  Procedure ShowName  Procedure ShowMembershipNo  Procedure AddAddress  Procedure AddName  Procedure AddMembershipNo  Procedure AmendAddress  Procedure AmendName  Procedure AmendMembershipNo  End |
| 7c) | JuniorMember = Class (Member)  Private  DOB: Date  Public  Procedure AddDOB  Procedure ShowAge  Procedure AmendDOB  End |
| 8) | The method will need to be polymorphic to make sure that the correct version of the method is called i.e. the one for ContractEmployee instead of the one for Employee.  Employee = Class  Private  Payroll  Surname  Forename  Address  MonthlyPaymentRecordsForCurrentYear  Public  AddEmployeeDetails  AmendEmployeeDetails  CalculateMonthlyPay Virtual  PrintPaySlip  End  SalariedEmployee = Class (Employee)  Private  AnnualSalary  Public  End  ContractEmployee = Class (Employee)  Private  HourlyRate  Public  CalculateMonthlyPay Override  End |
| 9a) | Membership number, forename, surname, address, telephone number, DOB, date joined, etc… |
| 9b) | PatientID, DOB, forename, surname, address, telephone number, date of last appointment, allergies, etc… |
| 10a) | Book, member, adult member, junior member |
| 10b) | Flight, Customer, Plane |
| 10c) | Room, Customer |

Chapter 1.2.3(2) Object-oriented programming

|  |  |
| --- | --- |
| 1) | The page number of the 36th page in the book aBook. |
| 2) | Composition |
| 3a) | Composition |
| 3b) | Dependency |
| 4a) | Dependency |
| 4b) | Inheritance |
| 4c) | Inheritance |
| 4d) | Composition |
| 5) | Aggregation |
| 6) | Aggregation |

Chapter 1.2.3(3) Object-oriented programming

|  |  |
| --- | --- |
| 1) | Time saving – as there is no need to write all the components as some are prewritten this saves development time.  Reduced error checking code – the access to an object’s state and internal code is highly-controlled, it can only be accessed through the object’s interface; this results in less code that needs to be checked for errors.  Code efficiency – the components are likely to have been developed by people who are experts in their areas meaning they will write more efficient code.  Reliability – the components will have been very thoroughly tested and written by experts and so are less likely to contain bugs. |

Chapter 1.2.3(4) Object-oriented programming

|  |  |
| --- | --- |
| 1) | Encapsulate what varies.  Favour composition over inheritance  Program to interfaces, not implementation |
| 2a) | It means that if new types of tool are added there is no need to rewrite any of the character class. |
| 2b) | Character  Tool  Master craftsman  Apprentice  Journeyman  Saw  Lockpick  Screwdriver  Hammer |
| 3) | Programming to an abstraction and not an implementation means that the program is not coupled to a specific implementation, the way it has been implemented can be changed without the programs using it having to be changed/rewritten as they only access the interface.  An interface cannot declare a constructor because all code in an interface is abstract and a constructor cannot be abstract.  An interface cannot have attributes for object state as an interface is used to state what methods need to be implemented, not how they are implemented or what data they may use.  An interface can’t do anything without an implementing class because the interface just specifies what methods there need to be – they have not actually been created, the class then contains the code for the methods specified in the interface. Without the implementing class there is no code associated with the methods specified in the interface. |
| 4) | Favouring composition over inheritance means that there will be less code duplication. |

Chapter 1.2.3(5) Object-oriented programming

|  |  |
| --- | --- |
| 1) | Public modifier means that the attribute or method is accessible/visible outside of its module.  Private modifier means that the attribute or method is not accessible/visible outside of its module.  Protected modifier means that the attribute or method is only accessible/visible outside of its module except by subclasses. |
| 2) | A static method is a method attached to the class itself and not to a particular instance of the class (object). A virtual method is one where calls are not resolved immediately by the compiler and instead the actual method to execute is determined at runtime though late binding – the actual class type of the object referenced by the variable is used to determine which method will be called. An abstract method is one which has been declared but not implemented, inheriting classes must implement the abstract method. |

Chapter 1.2.3(6) Object-oriented programming

|  |  |
| --- | --- |
| 1a) | Inheritance – an empty triangular arrowhead. |
| 1b) | Composition – a filled-in diamond arrowhead. |
| 1c) | Aggregation – an empty diamond arrowhead. |
| 1d) | Association – an arrowhead. |
| 2) |  |
| 3) |  |
| 4) |  |
| 5) |  |
| 6) | The right-most box at the bottom of the diagram is antibiotics. This is because is inherits from Particle (see below) for how we know which box is the Particle class.  The other three boxes on the bottom row are White blood cells, Bacteria, Resistant bacteria (order of the three does not matter). This is the case as these three are labelled as being subclasses that inherit from the box above them.  The box these three inherit from should be labelled Cell as this is the class the three should inherit from.  The box on the 2nd row should be labelled particle as a Cell is a type of Particle and the box labelled Cell inherits from this.  The top box should be labelled Environment as an Environment contains up to 10,000 particles. |
| 7) | The top box is School because a school is composed of a number of departments.  The box below this is Department as there are other classes being derived from it.  The two boxes below this are Computer Science and Maths as they inherit from Department.  The box at the bottom of the diagram is Teacher as there is shared aggregation with the Computer Science and Maths departments being able to both have the same teacher. |
| 8) | The top box of the diagram with the inheritance relationship is TForm as this is the base class from which TForm1 inherits.  The box below this is TForm1.  The top box of the other diagram is TApplication as this contains a TForm1 (which is the box below). TForm1 contains TEdit(s) and TButton so these are the two labels for the boxes at the bottom of the diagram. |
| 9) | FuelLevel needs to be made protected instead of private in the TAutomobile class. |

2 Fundamentals of Data structures

2.1 Data structures

Chapter 2.1.1 Data structures

|  |  |
| --- | --- |
| 1) | A data structure is a named collection of variables which are connected in different ways. |
| 2) | An array of records. A record will store all the data information about an order and the orders can then be stored in consecutive locations in the array. |

Chapter 2.1.2 Single- and multi-dimensional arrays (or equivalent)

|  |  |
| --- | --- |
| 1) | It means that one variable can be used instead of 10,000. |
| 2) | For Count 🡨 0 To 9  Output Height[Count]  EndFor |
| 3) | For Ch 🡨 'A' To 'Z'  Output LetterCount[Ch]  Endfor |
| 4) | Identification number |
| 5) | Extraction is the operation to get a value stored in an array e.g. Output Arr[2] will extract the value stored in position 2 in the array Arr and then display it on the screen.  Storing is the process of placing a value into a particular position in the array (assignment) e.g. Arr[3] 🡨 12 will store the integer value 12 in position 3 in the array Arr. |
| 6) | An array index can be thought of as an address as it is a number that specifies a location within an array. |
| 7a) | 9 |
| 7b) | Output Row[0] |
| 8a) | The variable i starts with a value of 0 and iterates over all the values between 0 and 9 (inclusive). On each iteration it will output the value in the ith position in the array Row. This means that it will print all the values store in the Row array. |
| 8b) | 10 |
| 9) | i 🡨 0  Repeat  Output Row[i]  i 🡨 i + 1  Until i = 10 |
| 10) | 2304 |
| 11) | Start indexing at 0: the index can be used as an offset when added to the first physical memory location of the array. |
| 12) | A vector is a one-dimensional array to which operations such as multiply can be applied to the array as a whole e.g. multiplying the vector represented by the one-dimensional array [3, 6, 7, 2] by 5 would give a new vector [15, 30, 35, 10]. |
| 13) | A matrix is a two-dimensional array to which operations such as multiply can be applied to the array as a whole e.g. multiplying the matrix ArrayTwoD by 5 would give:  [[15, 30, 35]  [20, 40, 5]  [45, 15, 40]] |
| 14) | An n-dimensional array is an array of n dimensions, where n is an integer greater than 0. |

Chapter 2.1.3 Fields, records and files

|  |  |
| --- | --- |
| 1) | A file is a data structure for storing data. Files use secondary storage unlike variables which are in RAM. |
| 2) | Text files are files whose contents are sequences of characters organized on a line-by-line basis. |
| 3) | A binary file is any file that is not a text file. |

Chapter 2.1.4a Abstract data types/data structures

|  |  |
| --- | --- |
| 1) | An abstract data type is a collection of data and a set of operations defined on the collection. |
| 2) | A data structure is an implementation of an ADT. |
| 3) | It creates an implementation-independent view of the data; this means that the implementation of the ADT can be changed without the need to change any programs that are using the data. |
| 4) | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | 18 | 9 | 27 | 12 | 4 | 35 |  | |  |  |  | Front |  | Rear |  | |
| 5) | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | 18 | 9 | 27 | 12 | 4 | 35 |  | |  |  |  |  |  |  |  |   Front = -1  Rear = -1  These values have been chosen as the queue is empty; the value of -1 is outside the index range of the array. |
| 6) | Because the item that has been in the queue the longest is the first item that will be removed from the queue. |
| 7) | |  |  | | --- | --- | |  |  | |  |  | | Top | 67 | |  | 16 | | Base | 3 | |
| 8) | The most recently added item will be the first item to be removed. |
| 9) | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | | Top | G |  |  |  | |  | F |  |  |  | |  | E |  |  |  | |  | D |  |  |  | |  | C |  |  |  | |  | B |  |  |  | | Base | A |  | Base/Top | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | |  | G |  |  |  | | Top | F |  |  |  | |  | E |  |  |  | |  | D |  |  |  | |  | C |  |  |  | |  | B |  | Top | G | | Base | A |  | Base | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | |  | G |  |  |  | |  | F |  |  |  | | Top | E |  |  |  | |  | D |  |  |  | |  | C |  | Top | F | |  | B |  |  | G | | Base | A |  | Base | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | |  | G |  |  |  | |  | F |  |  |  | |  | E |  |  |  | | Top | D |  | Top | E | |  | C |  |  | F | |  | B |  |  | G | | Base | A |  | Base | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | |  | G |  |  |  | |  | F |  |  |  | |  | E |  | Top | D | |  | D |  |  | E | | Top | C |  |  | F | |  | B |  |  | G | | Base | A |  | Base | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | |  | G |  |  |  | |  | F |  | Top | C | |  | E |  |  | D | |  | D |  |  | E | |  | C |  |  | F | | Top | B |  |  | G | | Base | A |  | Base | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  |  | |  | G |  | Top | B | |  | F |  |  | C | |  | E |  |  | D | |  | D |  |  | E | |  | C |  |  | F | |  | B |  |  | G | | Top/Base | A |  | Base | H |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | H |  |  | A | |  | G |  |  | B | |  | F |  |  | C | |  | E |  |  | D | |  | D |  |  | E | |  | C |  |  | F | |  | B |  |  | G | |  | A |  | Base | H |   Top/Base = -1 |
| 10) | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | | Rear |  |  |  |  | Front |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: B | | Rear |  |  |  | Front |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: | | Rear |  |  | Front |  |  | D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: | | Rear |  | Front |  |  |  | E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: | | Rear | Front |  |  |  |  | F  E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: | | Rear  Front |  |  |  |  |  | A  F  E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: | | Rear = -1  Front = -1 |  |  |  |  |  | C  A  F  E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | C | A | F | E | D | B | Stack: | | Rear  Front |  |  |  |  |  | A  F  E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | A | C | F | E | D | B | Stack: | | Rear | Front |  |  |  |  | F  E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | F | A | C | E | D | B | Stack: | | Rear |  | Front |  |  |  | E  D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | E | F | A | C | D | B | Stack: | | Rear |  |  | Front |  |  | D  B |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | D | E | F | A | C | B | Stack: | | Rear |  |  |  | Front |  | B | | B | D | E | F | A | C | Stack: | | Rear |  |  |  |  | Front |  | |
| 11a) | 200 |
| b) | 199 |
| c) | 0 |
| d) | -1 (or any other number outside of the range 0 to 199 |
| 12) | Stack full error / the stack will overflow i.e. there will be insufficient space in the stack for all the stack frames. |
| 13) | It will take up a lot of space on the stack (which could cause a stack full error). |
| 14) | Push 5  Push 3  Push 6  Pop 6  Pop 3  Push 18  Pop 18  Pop 5  Push 23  Pop 23 |
| 15a) | Push 5  Push 3  Pop 3  Pop 5  Push 8  Push 6  Push 4  Pop 4  Pop 6  Push 10  Pop 10  Pop 8  Push 180  Pop 180 |
| 15b) | Push 5  Push 2  Pop 2  Pop 5  Push 25  Pop 25 |
| 15c) | Push 3  Push 5  Push 2  Pop 2  Pop 5  Push 25  Pop 25  Pop 3  Push 28  Pop 28 |
| 15d) | Push 5  Push 3  Pop 3  Pop 5  Push 8  Push 6  Push 4  Pop 4  Pop 6  Push 10  Pop 10  Pop 8  Push 0.8  Pop 0.8 |
| 16) | Award mark for labelling one of 1, 2, 3 or 4 as a node/vertex.  Award mark for labelling one of the line between 1 and 2, the line between 2 and 3 or the line between 2 and 4 as an edge. |
| 17) | 38  130  160  56  184  144 |
| 18) | Cardigan  Synod Inn  Newcastle  Emlyn  Aberaeron  New Quay  Lampeter |
| 19) |  |
| 20) |  |
| 21) |  |
| 22) | |  |  |  |  | | --- | --- | --- | --- | |  | Data | Left | Right | | 1 | + | 2 | 3 | | 2 | \* | 4 | 5 | | 3 | \* | 6 | 7 | | 4 | 5 | 0 | 0 | | 5 | Y | 0 | 0 | | 6 | 4 | 0 | 0 | | 7 | Z | 0 | 0 | |
| 23) | A hash table is a table of informational elements organised on a row-by-row basis together with a hash function and table access procedures. The hash function is applied to a record’s key to generate a position in the table where the record is to inserted / where the record can be found.  It would be preferred to a linear list in this example because applying the hash function to the English word will directly identify the correct location in the table (time unaffected by size of table) whereas with a linear list the time taken to find the record will depend on the size of the list. |
| 24) | {‘0’:48, ‘1’:49, ‘2’:50, ‘3’:51, ‘4’:52, ‘5’:53, ‘6’:54, ‘7’:55, ‘8’:56, ‘9’:57} |
| 25) | (17, 27, 40, 58) |
| 26) | 1.16 + 2.25 + 4.36 +9.49 = 651 |
| 27a) | (0,0,0,11,13,15) |
| 27b) | (0,0,0,6,7.5,9) |
| 27c) | 122 |
| 28) | VB = (0,0,1,0,1,1,1,1)  When Alice and Bob’s vectors have the same value in a position then the value of that position in the secret key is 0.  When Alice and Bob’s vectors have different values in a position then the value of that position in the secret key is 1. |
| 29) | One way of doing it would be to still use addition over GF(2). Add the vectors for the three people to obtain the secret key vector.  0+0+0 = 0  0+0+1 = 1  0+1+0 = 1  0+1+1 = 0  1+0+0 = 1  1+0+1 = 0  1+1+0 = 0  1+1+1 = 1 |
| 30a) | 1.1+1.0+1.0+1.1+1.1+1.1+1.0 = 1+0+0+1+1+1+0  Dot product: 0 |
| 30b) | 1.1+1.0+1.0+1.1+1.1+1.1+1.1 = 1+0+0+1+1+1+1  Dot product: 1 |
| 30c) | Calculate the dot product of the vector with a 7-vector consisting of just 1s. The result will be the eight bit that should be appended. |

Chapter 2.1.4b Abstract data types/data structures

|  |  |
| --- | --- |
| 1) | Static data structures, e.g. fixed size arrays, have an amount of memory allocated to them; once allocated the amount of memory cannot change at run time.  Dynamic data structures, e.g. lists, can grow or shrink at run time to use the exact amount of memory required to store the data. |
| 2a) | When the number of data elements to be stored is known. |
| 2b) | When the number of data elements will change at run time. |
| 3) | Access time for a static data structure is independent of the order in which elements are accessed. They also don’t have to use memory to store pointers.  Dynamic data structures can grow and shrink at run time therefore avoiding wasted space. |

2.2 Queues

Chapter 2.2.1 Queues

|  |  |
| --- | --- |
| 1) | Instead of setting F to 0 when trying to add to an empty queue it should be set to 1.  Instead of setting R to 0 when trying to add to an empty queue it should be set to 1.  Instead of comparing QueueSize to R + 1 it should be compared to R. |
| 2a) | F = 0  R = 4 |
| 2b) | F = 3  R = 4 |
| 2c) | F = 3  R = 1 |
| 2d) | F = 1  R = 1 |
| 3) | Priority queue. |

2.3 Stacks

Chapter 2.3.1 Stacks

|  |  |
| --- | --- |
| 1) | 5 |
| 2) | -1 |
| 3ai) | 7 |
| 3aii) | 4 |
| 3b) | 6 |
| 3c) | True |
| 3d) | 2 |
| 3ei) | False |
| 3eii) | True |

2.4 Graphs

Chapter 2.4.1 Graphs

|  |  |
| --- | --- |
| 1) |  |
| 2) |  |
| 3) |  |
| 4) |  |
| 5a) |  |
| 5b) | 4 |
| 6) | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **A** | **B** | **C** | **D** | | **A** | 0 | 1 | 0 | 1 | | **B** | 1 | 0 | 1 | 1 | | **C** | 0 | 1 | 0 | 0 | | **D** | 1 | 1 | 0 | 0 | |
| 7) | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **1** | **2** | **3** | **4** | | **1** | 0 | 1 | 0 | 0 | | **2** | 1 | 0 | 1 | 1 | | **3** | 0 | 1 | 0 | 0 | | **4** | 0 | 1 | 0 | 0 |  |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |
| 8) | |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | | **1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **2** | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **3** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | **4** | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | | **5** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | | **6** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | | **7** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | **8** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | **9** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | **12** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 9) | A B,D  B A, C, D  C B  D A,B |
| 10) |  |
| 11) |  |
| 12) | In a social network application there will be many people (nodes) but each person will only know (be connected to) a small number of the other people in the social network. Therefore the graph that represents the social network will be quite sparse making an adjacency list likely to be the most suitable choice. |
| 13) | Adjacent matrix more appropriate when there are a large number of edges in a graph (i.e. graph is well-connected).  Matrix often more appropriate when edges frequently need to be added or deleted from a graph. |

2.5 Trees

Chapter 2.5.1 Trees (including binary trees)

|  |  |
| --- | --- |
| 1) | CBAEF  CBADGEF  CBADGEADGEF |
| 2) | a, c and d are trees; b and e are not trees |
| 3) | The theorem holds true for the trees shown. |
| 4) | H H H  | | |  H – C – C – C – H  | | |  H | H  |  H – C – H  |  H |
| 5) | Internal node  Branch  Leaf node  Root |
| 6) | a) E  b) C  c) E, A  d) B, C, D, E, F, G  e) E, D |
| 7) | A rooted tree in which each node is the parent of at most two other nodes is known as a binary tree.  Alternatively:  A binary tree is an abstract data type defining a finite set of elements. The set is either empty or is partitioned into three subsets. The first subset contains a single element called the root of the tree. The other two subsets are themselves binary trees, called the left and right sub-trees of the original tree. |
| 8) | d) is the only binary tree. b) is a rooted tree but not a binary tree. |
| 9) | Left sub-tree    Right sub-tree |
| 10) | Left sub-tree of left sub-tree |
| 11) |  |
| 12) |  |
| 13a) | 3 |
| 13b) | 30, 20, 25 |
| 14) | pull  Strong  oxen  well |
| 15) | y  \*  3  :=  y |

|  |  |
| --- | --- |
| 16) | h t  H T  h t h t  HH HT TH TT  h t h t h t h t  HHH HHT HTH HTT THH THT TTH TTT  h t h t h t h t h t h t h t h t  HHHH HHHT HHTH HHTT HTHH HTHT HTTH HTTT THHH THHT THTH THTT TTHH TTHT TTTH TTTT |
| 17) | 9 |
| 18) | 123<456 123>456 123=456  123 456 78  1<3 1=3 1>3 4<6 4=6 4>6 7<8 7>8  1 2 3 4 5 6 7 8 |
| 19) | 16 |
| 20a) | \*  20b)  \*  +  +  /  +  z  y  2  x  4  5  - +  x 6 y 8 |
| 21) | 1  1.1 1.2 1.3  1.1.1 1.1.2 1.3.1 1.3.2  1.1.1.1 1.1.1.2 |

|  |  |
| --- | --- |
| 22 | 62 |
| 23a) | 65354 |
| 23b) | 65355 |
| 23c) | 15 |

2.6 Hash tables

Chapter 2.6.1 Hash tables

|  |  |
| --- | --- |
| 1a | 0 |
| 1b | 3 |
| 1c | 3 |
| 2a | 509 |
| 2b | 80 |
| 2c | 20 |
| 2d | 20 |
| P1 | Answer written using Python 3  def H(key):  summ = 0  for i in range(len(key)):  summ += ord(key[i])\*\*2  hash = summ % 523  return (hash)  words = []  for count in range(523):  pair = ["-1",""]  words.append(pair)  location = H("PEN")  words[location][0] = "PEN"  words[location][1] = "PLUME"  location = H("CAT")  words[location][0] = "CAT"  words[location][1] = "CHAT"  location = H("NOW")  words[location][0] = "NOW"  words[location][1] = "MAINTENANT" |
|  | Answer written using Python 3  def H(key):  summ = 0  for i in range(len(key)):  summ += ord(key[i])\*\*2  hash = summ % 523  return (hash)  words = []  for count in range(523):  pair = ["-1",""]  words.append(pair)  location = H("PEN")  words[location][0] = "PEN"  words[location][1] = "PLUME"  location = H("CAT")  words[location][0] = "CAT"  words[location][1] = "CHAT"  location = H("NOW")  words[location][0] = "NOW"  words[location][1] = "MAINTENANT"  again = ""  while again != "N":  key = input("Enter an English word to search for:")  frenchWord = words[H(key)][1]  print ("The French equivalent of the English word is:",frenchWord)  again = input("Look for another word (enter N to stop)?") |
| 3 | |  |  |  | | --- | --- | --- | | **ULN** | **ULN Mod 7** | **ULN Mod 11** | | 24567805 | 3 | 9 | | 34567876 | 0 | 2 | | 64156906 | 2 | 0 | | 74432167 | 5 | 7 | | 90002789 | 2 | 8 | | 90002985 | 2 | 6 | | Table 2.6.6 | | | |
| 4 | |  |  |  |  | | --- | --- | --- | --- | |  | **ULN** | **Forename** | **Surname** | | 0 | 34567876 |  |  | | 1 |  |  |  | | 2 | 64156906 |  |  | | 3 | 24567805 |  |  | | 4 | 90002789 |  |  | | 5 | 74432167 |  |  | | 6 | 90002985 |  |  | | Table 2.6.7 Hash table | | | | |
| 5 | |  |  |  |  | | --- | --- | --- | --- | |  | **ULN** | **Forename** | **Surname** | | 0 | 64156906 |  |  | | 1 |  |  |  | | 2 | 34567876 |  |  | | 3 |  |  |  | | 4 |  |  |  | | 5 |  |  |  | | 6 | 90002985 |  |  | | 7 | 74432167 |  |  | | 8 | 90002789 |  |  | | 9 | 24567805 |  |  | | 10 |  |  |  | | Table 2.6.8 Hash table | | | | |
| 6 | The hash function would be applied to the key:  24567805 Mod 11 = 9  There is a record in address 9 and the ULN of that record is the one being looked for so the forename and surname of the record in address 9 would be retrieved. |
| 7 | The hash function would be applied to the key:  24567805 Mod 7 = 3  There is a record in address 3 and the ULN of that record is the one being looked for so the forename and surname of the record in address 3 would be retrieved. |
| 8 | More than one key field value could map onto the hash value so the record stored in the address obtained when applying the hash function to the key field value may not be the record with that key field value. If the key field value was not stored in the record it would not be possible to detect that this issue had occurred. |
| 9 | |  |  |  |  | | --- | --- | --- | --- | |  | **Variable name** |  |  | | 0 | OVERTIME | TAXRATE |  |  |  |  |  |  | | --- | --- | --- | --- | | 1 | P | URL |  |  |  |  |  |  | | --- | --- | --- | --- | | 2 | GENDER |  |  |  |  |  |  |  | | --- | --- | --- | --- | | 3 | CHECK | MAIN |  | | 4 | INDEX | N |  | |
| 10a | 509 |
| 10b | 509 |
| 10c | The two keys have generated the same address so a collision has occurred. |
| 10d | Linear rehashing – the 2nd record is stored in the next free address (i.e. the next unused address after 509, though it should wrap round to the start, memory address 0, if no free memory location can be found after 509).  Closed addressing – a linked list is associated with that memory address. A pointer is used to point to the start of the linked list and the 2nd record is stored in the linked list. |
| 11 | Because if the entry at the original hash value table index or any of the rehash value table entries are deleted, and the deleted entry remains empty, searching can be stopped prematurely before all potential matching entries have been examined. |
| 12a | n+1 |
| 12b | You know that socks are stored in that drawer (equivalent to address in table) but there are multiple socks that are supposed to be stored in the same drawer (collisions) so a pile (equivalent to linked list) is used to store the socks. |
| 13a | There would be a very large amount of wasted space as there would never be more than 500 items but memory space for 89999999 would be used to represent the table. |
| 13b | Far less memory would be needed for the table as a hash function could be used to map the 500 actual ULNs used onto a much smaller address range. |
| 14a | Quicker searching  Quicker/Easier to insert a new record in the correct place (no need to move other records around to fit the new record in) |
| 14bi | A large number of collisions have occurred meaning that a significant number of records are not actually stored in the address generated by applying the hash function to their key value.  If the record is not in the location generated by the hash function then the following memory locations have to be checked to see if the record is in one of those.  If this starts happening frequently then search time will start to increase. |
| 14bii | Rehash the table using a different hash function that will generate fewer collisions. Sometimes a better hash function can be used while keeping the same amount of addresses that can be mapped onto by the hashing function, sometimes a hash function that generates a wider range of potential addresses has to be used (and more memory allocated to the hash table). |
| 15 | More than one hash key may hash to the same address. The hash key must be stored in order to check for a possible clash. |

2.7 Dictionaries

2.7.1 Dictionaries

|  |  |
| --- | --- |
| 1) | In this case the key-value pairs will be character-code. The pairs associated with all the non-control characters will be inserted into the dictionary with their associated values (character codes) e.g. insert(“A”, 65) to add the key (character “A”) to the dictionary along with its value (the ASCII code 65). |

2.8 Vectors

2.8.1 Vectors

|  |  |
| --- | --- |
| 1) | (18, 24, 31) |
| 2) | (3, 9, 15, 27) |
| 3) | 6.0 + 30.0 -7.0 = 29.0 |
| 4) | (0.8, 1.6) + (11.2, 12.8) = (12, 14.4) |
| 5) | Length of vector A is 2.00004 to 5dp  Length of vector B is 1  Dot product of A and B is (1.0, 1.7321).(1.0,0) = 1.0  1.0 / (2.00004\*1) = 0.49999 to 5dp  So angle is 60.0 degrees to 1dp |

3 Fundamentals of algorithms

3.1 Graph-traversal

3.1.1 Simple graph-traversal algorithms

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1) | |  |  |  |  | | --- | --- | --- | --- | | **CurrentVertex** | **Neighbours** | **Visited vertices** | **Queue Q**  **Front Rear** | | A | B, C | A | A | | A | B, C | A, B | B | | A | B, C | A, B, C | B, C | | B | A | A, B, C | C | | C | A, D, E | A, B, C |  | | C | A, D, E | A, B, C, D | D | | C | A, D, E | A, B, C, D, E | D, E | | D | C, F, G | A, B, C, D, E | E | | D | C, F, G | A, B, C, D, E, F | E, F | | D | C, F, G | A, B, C, D, E, F, G | E, F, G | | E | C, G | A, B, C, D, E, F, G | F, G | | F | D, G | A, B, C, D, E, F, G | G | | G | D, E, F | A, B, C, D, E, F, G |  | |
| 2) | A, B, C, D, E, F, G |
| 3) | |  |  |  |  | | --- | --- | --- | --- | | **CurrentVertex** | **Neighbours** | **Visited vertices** | **Stack S**  **Base Top** | | A | B, C | A | A | | B | A | A, B | A, B | | A | B, C | A, B, C | A | | C | D, E | A, B, C, D | A, C | | D | F, G | A, B, C, D, F | A, C, D | | F | D, G | A, B, C, D, F | A, C, D, F | | G | D, E, F | A, B, C, D, F, G | A, C, D, F, G | | G | D, E, F | A, B, C, D, F, G, E | A, C, D, F, G, E | | E | C, D, G | A, B, C, D, F, G, E | A, C, D, F, G, E | | G | D, E, F | A, B, C, D, F, G, E | A, C, D, F, G | | F | D, G | A, B, C, D, F, G, E | A, C, D, F | | D | F, G | A, B, C, D, F, G, E | A, C, D | | C | D, E | A, B, C, D, F, G, E | A, C | | A | D,E | A, B, C, D, F, G, E | A | |
| 4) | A, B, C, D, F, G, E |
| 5a) | Applications where the shortest path between two nodes needs to be found. |
| 5b) | Navigating a maze |

3.2 Tree-traversal

3.2.1 Simple tree-traversal algorithms

|  |  |
| --- | --- |
| 1a) | +3↑52 |
| 1b) | 3+5↑2 |
| 1c) | 352↑+ |
| 2a) | \*+53+64 |
| 2b) | 5+3\*6+4 |
| 2c) | 53+64+\* |
| 3a) | +\*-AB+CD/EG |
| 3b) | A-B\*C+D+E/G |
| 3c) | AB-CD+\*EG/+ |
| 4a) | Pre-order traversals can be used to make a copy of a binary tree. |
| 4b) | In-order traversals can be used to produce the data in an ordered binary tree in ascending order. |
| 4c) | Post-order traversals can be used to convert an infix expression into Reverse Polish Notation. |

3.3 Reverse Polish

3.3.1 Reverse Polish – infix transformations

|  |  |
| --- | --- |
| 1a) | 5 3 + 6 4 + \* |
| 1b) | 5 2 ↑ |
| 1c) | 3 5 2 ↑ + |
| 1d) | 5 3 + 6 4 + / |
| 2a) | 4 \* (7 + 2) - 3 |
| 2b) | (6 + 4↑2 )\* 8 |
| 2c) | (3 \* (5 – 2))↑6 / 5 |
| 3) | RPN is used because it a parenthesis-free notation i.e. no need for brackets. |
| 4) | Interpreters for virtual stack machines are much simpler to build than interpreters for register machines. RPN uses a stack for evaluating expressions which means it works well with a virtual stack machine interpreter.  The same advantage is used when making compilers for virtual stack machines. These compilers are much simpler enabling them to fit onto lower-specification machines. |

3.4 Searching algorithms

3.4.1 Linear search, 3.4.2 Binary search, 3.4.3 Binary tree search

|  |  |
| --- | --- |
| 1a) | 26.5 |
| 1b) | 52 |
| 2) | 26 |
| 3) | 5 |
| 4) | Linear search means to look at each item in a collection in turn until either the item is found or the end of the collection is reached. |
| 5) | Binary search keeps chopping a list into smaller and smaller lists to search until the item is found or the list cannot be divided anymore. |
| 6) | The item must be in the list.  The list must be ordered. |
| 7) | Because it takes fewer comparisons (in the worst case and in the average case) to find an item in a list. |
| 8a) | No |
| 8b) | Yes |
| 9) | At most it will take 21 comparisons to guess the number and the average time taken will be the same as this, so most people will not get the correct number within 10 guesses. To be likely to make a profit the stall need it to be more likely than not that the first 11 people will not guess the number correctly.  Overall, it is likely that the stall will make a profit – particularly as a lot of people won’t use a binary search algorithm to determine their guesses. |
| 10a) | Best case  Average case |
| 10b) | Because it is often important to know what the maximum amount of time taken could be, e.g. safety-critical systems; average time is difficult to measure accurately as it performance on random data sets can be very variable. |
| 11) | Because if there are n items in the list then it could take n comparisons to find an item in the list (if it is the last item in the list or if it not in the list). |
| 12) | Because the maximum number of comparisons goes up by 1 each time the size of the list doubles. |
| 13) | Because in a balanced tree with n nodes there will be at most log2n + 1 levels in the binary tree (and same maximum number of comparisons therefore needed). |

3.5 Sorting algorithms

3.5.1 Bubble sort

|  |  |
| --- | --- |
| 1) | F E E E E EE F D D D DD D F C C CC C C F B BB B B B F AA A A A A FE D D D DD E C C CC C E B BB B B E AA A A A EF F F F FD C C CC D B BB B D AA A A DE E E EF F F FC B BB C AA A CD D DE E EF F FB AA BC CD DE EF F |
| 2) | Change > to <. |

3.5.2 Merge sort

|  |  |
| --- | --- |
| 1) | To merge sort a list the list is split into two lists and the merge sort algorithm is then applied to each of the half lists. This process continues until a base case is reached (list with only two elements) when the elements in the list are compared with each other and swapped if in the wrong order.  When the base case has been reached each pair of sublists are merged together – a process that combines two ordered lists into one, larger, ordered list containing all the elements from the two lists. |
| 2) | MergeSort(CU, U, 0, 4) TMLK TMLK  MergeSort(B, A, 0, 2) TMLK TMLK  MergeSort(B, A, 2, 4) MTLK TMLK  Merge(A, B, 0, 2, 4) MTKL TMLK  Final state of arrays KLMT MTKL |
| 3) | MergeSort(CU, U, 0, 8)  MergeSort(U, CU, 0, 4)  MergeSort(CU, U, 0, 2)  MergeSort(CU, U, 2, 4)  Merge(U, CU, 0, 2, 4)  MergeSort(U, CU, 4, 8)  MergeSort(CU, U, 4, 6)  MergeSort(CU, U, 6, 8)  Merge(U, CU, 4, 6, 8)  Merge(CU, U, 0, 4, 8) |

3.6 Optimisation algorithms

3.6.1 Dijkstra’s shortest path algorithm

To be revisited because question 2 changed.

1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Itera-  tion | S | U | w | D[0] | D[1] | D[2] | D[3] | D[4] | P[0] | P[1] | P[2] | P[3] | P[4] |
| Initial | {} | {0,1, 2, 3, 4} | - | 0 | ∞ | ∞ | ∞ | ∞ | 0 |  |  |  |  |
| 1 | {0} | {1, 2, 3, 4} | 0 | 0 | 5 | ∞ | 10 | ∞ | 0 | 0 | 0 | 0 | 0 |
| 2 | {0,1} | {2, 3, 4} | 1 | 0 | 5 | 7 | 8 | 14 | 0 | 0 | 1 | 1 | 1 |
| 3 | {0,1,2} | {3, 4} | 2 | 0 | 5 | 7 | 8 | 13 | 0 | 0 | 1 | 1 | 2 |
| 4 | {0,1,2,3} | {4} | 3 | 0 | 5 | 7 | 8 | 9 | 0 | 0 | 1 | 1 | 3 |
| 5 | {0,1,2,3,4} | {} | 4 | 0 | 5 | 7 | 8 | 9 | 0 | 0 | 1 | 1 | 3 |

2)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Itera-  tion | S | V | w | D[0] | D[1] | D[2] | D[3] | D[4] | P[0] | P[1] | P[2] | P[3] | P[4] |
| 1 | {0} | {1, 2, 3, 4} | 0 | 0 | 60 | 200 | 170 | ∞ | 0 | 0 | 0 | 0 | -1 |
| 2 | {0,1} | {2, 3, 4} | 1 | 0 | 60 | 180 | 170 | 260 | 0 | 0 | 1 | 0 | 1 |
| 3 | {0,1,2} | {3, 4} | 3 | 0 | 60 | 180 | 170 | 260 | 0 | 0 | 1 | 0 | 1 |
| 4 | {0,1,2,3} | {4} | 2 | 0 | 60 | 180 | 170 | 260 | 0 | 0 | 1 | 0 | 1 |
| 5 | {0,1,2,3,4} | {} | 4 | 0 | 60 | 180 | 170 | 260 | 0 | 0 | 1 | 0 | 1 |

3) Use Dijikstra’s algorithm. Represent the flights between the airports as a graph with the difference between the two times as a weight. When constructing the graph start with the origin airport and only add edges for flights which have a departure time after the calculated arrival time from the origin airport (using the start time as an initial value as “arrival time” for the origin airport). Then apply Dijikstra’s algorithm to find the shortest path from the origin airport.

4 Theory of computation

4.1 Abstraction and automation

Chapter 4.1.1 Problem-solving

|  |  |
| --- | --- |
| 1a) | If the wind blows from the North, then it snows. |
| 1b) | If not exercising enough, then get unfit. |
| 1c) | If leaves are turning brown on the trees, then it is autumn. |
| 1d) | If water is boiling, then it is 100°C. |
| 1e) | If you bought the goods in the last two weeks, then you can get a refund. |
| 2) | Major proposition: if it is cold, he wears a hat  Minor proposition: it is cold  Conclusion: he is wearing a hat |
| 3) | No, the proposition states that if it is cols he wears a hat but does not mean that there are no other times when he wears a hat. |
| 4) | The politician will lower taxes. |
| 5) | No, because taxes may have been lowered by the politician even if he did not get elected. |
| 6) | No |
| 7) | Ben has a valid network password.  If you have a valid network password then you can log into the school’s network. |
| 8) | |  |  |  | | --- | --- | --- | | **I am elected** | **I lowered taxes** | **If I am elected, I will lower taxes** | | True | True | True | | True | False | False | | False | True | True | | False | False | True |   The 2nd row corresponds to the politician breaking his promise. |
| 9) | P ⇒ Q  P  ∴Q  It is a valid argument.  The conclusion is true. |
| 10) | Because the major proposition is not true. |
| 11) | Yes |
| 12) | No, because the major proposition is not true. |
| 13) | If it snows today I can catch up on my homework. |
| 14) | If the train arrives early Jamin can call his friend John. |
| 15) | If I don’t watch the late night film I will wake up refreshed. |
| 16) | If Alex is allowed a TV in his room he will need extra tuition. |
| 17) | There are taxis at the airport. |
| 18) | Isla has her coat with her. |
| 19) | U – needs to be turned over to check there is an even number on the other side.  3 – needs to be turned over to check that there is not a vowel on the other side.  8 – does not need to be turned over as if there is not a vowel on the other side it does not disprove the proposition (nothing is stated about what is true for consonants).  L – does not need to be turned over as it does not matter if there is an even or odd number of the other side it will not help to prove or disprove the proposition. |
| 20) | False  True |
| 21) | Gerry drinks coffee. |
| 22) | Deemei has learned at least one programming language. |
| 23) | They are both lying (if the statement that one of them is lying is true). |

Chapter 4.1.2 Following and writing algorithms

|  |  |
| --- | --- |
| 1) | Product 🡨 Product + y  x 🡨 x + 1 |
| 2) | Answer 🡨 Answer \* x  Count 🡨 Count + 1 |
| 3) | m 🡨 1  While 2^m < N  m 🡨 m + 1  EndWhile  m 🡨 m – 1 |
| 4) | Sum 🡨 0  For Count 🡨 1 To n  Sum 🡨 Sum + Count  EndFor |
| 5) | Product 🡨 1  For Count 🡨 1 To n  Product 🡨 Product \* Count  EndFor |
| Tasks |  |
| 2) | |  |  |  |  | | --- | --- | --- | --- | | **Dividend x** | **Divisor y** | **Quotient q** | **Remainder r** | | 5 | 2 | 2 | 1 | | 6 | 3 | 2 | 0 | | 25 | 4 | 6 | 1 | | 36 | 6 | 6 | 0 | | 121 | 7 | 17 | 2 | | 23 | 3 | 7 | 2 | | 1 | 3 | 0 | 3 | | 5 | 10 | 0 | 10 | |
| 3) | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Iteration No** | **x** | **y** | **r** | **q** | **r > y** | | 0 | 6 | 3 | 6 | 0 | True | | 1 | 6 | 3 | 3 | 1 | False | |  |  |  |  |  |  | |
| 4a) | Yes |
| 4b) | No |
| 5) | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Iteration No** | **x** | **y** | **r** | **q** | **r > y** | | 0 | 6 | 0 | 6 | 0 | True | | 1 | 6 | 0 | 6 | 1 | True | | 2 | 6 | 0 | 6 | 2 | True | | 3 | 6 | 0 | 6 | 3 | True | | 4 | 6 | 0 | 6 | 4 | True | | 5 | 6 | 0 | 6 | 5 | True | |
| 6) | No |
| 7) | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Iteration No** | **x** | **y** | **r** | **q** | **r >= y** | | 0 | -2 | 1 | -2 | 0 | False | |
| 8) | Yes |
| 9) | No |

Chapter 4.1.3 Abstraction

|  |  |
| --- | --- |
| 1a) | Omitting unnecessary details. |
| 1b) | Grouping by common characteristics to arrive at a hierarchical relationship of the ‘is a kind of’ type. |
| 2) | Removing unnecessary details means that it is often easier to find a solution to the problem as it is easier to see the information that is important and to spot how to use the information to solve the problem. |

Chapter 4.1.4 Information hiding

|  |  |
| --- | --- |
| 1) |  |
| 2) | Information hiding is the process of hiding all details of an object that do not contribute to its essential characteristics. When developing software systems this is done via an interface where other modules interact with a module via its interface, they do not need to know the internal workings of the module.  If a software system wants to make use of Google Maps it can do this via an interface (the Google Maps API) and does not need to know how the Google Maps code works. |

Chapter 4.1.5 Procedural abstraction

|  |  |
| --- | --- |
| 1) | A formula is a whole class of computations – since each variable in the formula can be replaced with an infinite number of different values. A formula represents a computational method, a procedure. Such an abstraction is called a procedural abstraction, since the result of the abstraction is a procedure, or method. |

Chapter 4.1.6 Functional abstraction

|  |  |
| --- | --- |
| 1) | Functional abstraction is when the computation method actually used is hidden. The function is a black box – we put the necessary inputs into the box and the desired output is produced, there is no need to know what is going on inside the box to produce the output. |

Chapter 4.1.7 Data abstraction

|  |  |
| --- | --- |
| 1) | Details of how data are actually represented are hidden. |
| 2a) | It could be represented using an array of compound objects. Each object would store the necessary details about the book e.g. ISBN, author, title, etc. The books details would be stored in consecutive locations in the array. |
| 2b) | Add to stack (push)  Delete from stack (pop)  Check for empty stack  Check for full stack  Flush  Look at top item of stack (peek) |
| 2c) | Create an interface which contains headers for the subroutines available e.g. push. Put the code for the subroutines and the array in an implementation section that is not accessible except via the interface. |

Chapter 4.1.8 Problem abstraction/reduction

|  |  |
| --- | --- |
| 1) | If there are n people (assuming that no one shakes their own hand or shakes hands with a person more than once) then each person will shake hands with between 1 and n-1 people. It is not possible to assign each person a different number of handshakes because there is one less possible number of handshakes than there is number of people so at least two people must have done the same number of handshakes. |
| 2) | 3, because if you use the pigeonhole principle and the colours are equivalent to holes there are two holes (colours) so if you take three socks at least two of them must be put in the same hole (be of the same colour). |
| 3) |  |

Chapter 4.1.9 Decomposition

|  |  |
| --- | --- |
| 1) | Procedural decomposition is a top-down strategy in which big problems are broken into smaller problems. |

Chapter 4.1.10 Composition

No questions.

Chapter 4.1.11 Automation

1) Many possible answers

4.2 Regular languages

Chapter 4.2.1 Finite state machines (FSMs) with and without output

|  |  |
| --- | --- |
| 1) | a) Yes  b) No  c) No  d) Yes  e) No  f) Yes |
| 2) | a) Yes  b) No  c) Yes  d) No  e) No  f) No  g) No |
| 3) | 1  0  1  0  a  a or b  b |
| 4) | a) Yes  b) Yes  c) Yes  d) No  e) Yes  f) Yes  g) No |
| 5) | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Current state** | S0 | S0 | S1 | S1 | | **Input symbol** | a | b | a | b | | **Next state** | S1 | S0 | S1 | S0 | |
| 6) | |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Current state** | S0 | S0 | S1 | S1 | S2 | S2 | S3 | S3 | | **Input symbol** | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | **Next state** | S1 | S3 | S2 | S1 | S3 | S2 | S3 | S2 | |
| 7) | To be done |
| 8) | 000100 |
| 9a) | 00000000 |
| 9b) | S0 |
| 10) | 1|0  0|1  0|1  1|0 |
| 11a) | =0 |
| 11b) | =1 |
| 11c) | =1 |
| 11d) | =0c1 |

4.2 Regular languages

Chapter 4.2.2 Maths for regular expressions

|  |  |
| --- | --- |
| 1a) | {4, 6, 9, 2} *– note: order of elements not important* |
| 1b) | {1, 2, 7, 8, 6} *– note: order of elements not important* |
| 2) |  |
| 3) | 4, 9, 16, 25, 36 |
| 4) | A set whose elements can be counted off by the natural numbers up to a particular number. |
| 5) | A set is one that can be placed in one-to-one correspondence with the natural numbers. |
| 6) | The set of natural numbers  The set of rational numbers |
| 7a) | |4| |
| 7b) | |3| |
| 7c) | |0| |
| 8) | The set of allocations of lockers to students i.e. a set of ordered pairs indicating which student has which locker. |
| 9) | {(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)} |
| 10) | The set of possible journeys i.e. airline, source airport, destination airport. |
| 11) | The set of all (x, y) coordinates. |
| 12a) | True |
| 12b) | True |
| 12c) | False |
| 12d) | True |
| 12e) | True |
| 13a) | {2, 4, 1, 3} |
| 13b) | {1, 2, 3, 4} |
| 13c) | {2} |
| 13d) | {} |
| 13e) | {4, 8} |
| 13f) | {} |

Chapter 4.2.3 Regular expressions

|  |  |
| --- | --- |
| 1) | 001  010  011  100  101  110  111 |
| 2) | 000  010  101  111 |
| 3a) | Yes |
| 3b) | Yes |
| 3c) | No |
| 3d) | No |
| 4a) | Yes |
| 4b) | Yes |
| 4c) | Yes |
| 4d) | No |
| 4e) | No |
| 5) | (A|C|G|T)\* |
| 6) | (A|C|G|T|a|c|g|t)\* |
| 7a) | Identifier must start with a letter followed by any combination of letters, numbers and underscores (including none of these). |
| 7bi) | Yes |
| 7bii) | Yes |
| 7biii) | No |
| 7biv) | Yes |
| 8ai) | Yes |
| 8aii) | Yes |
| 8aiii) | No |
| 8aiv) | No |
| 8b) | 30/2/2016 |
| 9a) | The \ means escape the normal meaning of the following character. Therefore the + is treated as the symbol instead of the metasymbol. |
| 9b) | The first \ means that the second \ is not treated as having its normal meaning so it is not an escape character just a normal character. |
| 9c) | The \ preceeding the – means that the – is not a metacharacter; when the \ is not included the – is treated as a metacharacter. |
| 10a) | Yes. The + or – is optional and not included here. There are then one or more numeric digits (three). |
| 10b) | Yes. A – followed by one or more numeric digits (4). |
| 10c) | Yes. A + followed by one or more numeric digits (5). |
| 10d) | No. There must be at least one numeric digit. |
| 11a) | Yes. The only compulsory element according to the regular expression is one or more numeric characters (\d+) which this string contains. |
| 11b) | Yes. Has a – at start (optional character that is allowed) followed by one or more numeric digits then an optional . followed by one numeric digit. |
| 11c) | Yes. Has a + at start (optional character that is allowed) followed by one or more numeric digits then an optional . followed by one or more numeric digits. |
| 11d) | No. The letter d is not allowed as it does not match \d in the regular expression. |
| 12) | A sequence of characters followed by an @ symbol with a series of characters followed by a dot and a sequence of characters are the compulsory elements and are all present in the first string. The last sequence of characters contains another dot followed by a sequence of characters which is allowed.  The second string is not allowed as it contains characters like !,;,#,£ which are not allowed by the regular expression. Also there needs to be at least one . after the @ followed by some characters, and this is not present in the second string. |
| 13a) | Yes |
| 13b) | Yes |
| 13c) | No |
| 13d) | No |
| 14a) | No |
| 14b) | Yes |
| 14c) | Yes |
| 14d) | No |
| 15a) | Yes |
| 15b) | No |
| 15c) | Yes |
| 15d) | Yes |
| 16a) | No |
| 16b) | No |
| 16c) | Yes |
| 16d) | Yes |
| 17a) | Yes |
| 17b) | No |
| 17c) | No |
| 17d) | Yes |
| 17e) | No |
| 18) | li(c|s)en(c|s)e |
| 19) | GCG(CGG)+CTG |
| 20a) |  |
| 20b) |  |
| 21) | a(c|b+c)  or  ac|ab+c |

Chapter 4.2.4 Regular language

No questions.

4.3 Context-free languages

Chapter 4.3.1 Backus-Naur (BNF)/Syntax diagrams

|  |  |
| --- | --- |
| 1a) | Backs-Naur Form |
| 1b) | 3B – Yes  3HB – No |
| 2a) | 12 – Yes  35.45 – No  (6+7)x4 – Yes |
| 2b) | <expression>  / | \  <expression> x <expression>  | / | \  <number> ( <expression> )  | / | \  <digit> <expression> - <expression>  | | |  4 <number> <number>  | / \  <digit> <digit> <number>  | | |  7 1 <digit>  |  5 |
| 3a) | 3Loop – No  Test6 – Yes  Loop1Test – Yes |
| 3b) | <assignment\_statement>  / | \  <identifier> := <expression>  | \ / | \  <letter> <string> <expression> \* <expression>  | | | |  T <character> <identifier> <integer>  | | |  <digit> <letter> <digit>  | | |  1 R 6 |
| 4) | The rules of a language. |
| 5a) | <string> ::= b | a <string> | b <string> |
| 5b) | <string> ::= Ɛ | a <listofas> | b <listofbs> | ab <listofabs>  <listofas> ::= Ɛ | a <listofas>  <listofbs> ::= Ɛ | b <listofbs>  <listofabs> ::= Ɛ | ab <listofabs> |
| 6) | A language where there can be any number of open brackets - ( - but there must be the same number of closing brackets - ). This cannot be represented by a regular expression as these have no way of counting or using recursion. |
| 7a) | Syntax diagram |
| 7b) | MyCount3 – No  x : integer; - No  var x, y : integer; flag : boolean; - Yes |
| 7c) | <type> ::= integer | float | boolean | string  <identifier> ::= <letter> | <letter> <identifier>  <variable\_declaration> ::= var <variables>  <variables> ::= <list> : <type> ; | <variables><variables>  <list> ::= <identifier> | <identifier>,<identifier> |
| 8) |  |
| 9) |  |

4.4 Classification of algorithms

Chapter 4.4.1 Comparing algorithms

|  |  |
| --- | --- |
| 1) | The dominating term of an order of magnitude function. |
| 2a) | *N*  *N*2 + 6*N*  *N*2  N + 5*N*2 + 2*N*3  *N*3  *N*4 – 4 |
| 3) |  |
| 4) | C  A  B |
| 5) | C  A  B |
| 6) | f(N) = N |
| 7) | f(N) does not change with N |

Chapter 4.4.2 Maths for understanding Big-O notation

|  |  |
| --- | --- |
| 1) | {0, 1, 8, 27} |
| 2) | f(x) = 2x |
| 3) | 1 |
| 4) | It quadruples |
| 5) | It increases by 1 |
| 6) | log2x, 2x, 2x2, 2x3 |
| 7ai) | 16 |
| 7aii) | 1024 |
| 7aiii) | 18,446,744,073,709,551,616 |
| 7b) | f(x) = 2x |
| 8a) | 24 |
| 8b) | 362,880 |
| 8c) | 6,227,020,800 |

Chapter 4.4.3 Order of complexity

|  |  |
| --- | --- |
| 1) | Big O is a notation used to classify algorithms by how their computation time changes with the size of the input on which they operate. |
| 2) | O(log2n), O(n), O(n2), O(n3), O(2n) |
| 3a) | The time taken is some constant times the size of the input n. |
| 3b) | The time taken grows dramatically as the size of the input n increases. |
| 3c) | The time taken can be expressed as a polynomial in the size of the input n. |
| 4a) | O(n) |
| 4b) | O(1) |
| 5a) | O(2n) |
| 5b) | 29 |
| 6) | O(n2) |
| 7a) | 4 |
| 7b) | n is the number of boxes to be drawn  Drawing method – O(n)  Folding method – O(log2n) |

Chapter 4.4.4 Limits of computation

|  |  |
| --- | --- |
| 1) | Computers have a limited amount of memory and some problems may require more memory than the computer has.If an algorithm has exponential time complexity then, as the size of the input grows, it quickly becomes impossible to solve the problem in a reasonable amount of time. Increasing the number of processors used, or the speed of the processor, does not have any significant impact on this. |

Chapter 4.4.5 Classification of algorithmic problems

|  |  |
| --- | --- |
| 1a) | ABCDA or ADCBA (the two routes are mirrors of each other) |
| 1b) | 100! (or 100!/2 if mirrored routes not counted) |
| 1c) | Intractable – the time complexity is O(n!) which is worse than polynomial time. |
| 2a) | 1000! |
| 2b) | 1000!/10,000,000 seconds  or  4.02387260077093773543702433923 x 102650 |
| 2c) | Because it can’t be solved in polynomial time – each bit added causes the time taken to increase dramatically. |
| 3) | Because generating a password can be completed in linear time (less than polynomial time). The main operation is to generate a 1 or 0. If it takes a second to generate a single 1 or 0 then it will take two seconds to generate two values, etc… Generating a password has time complexity of O(n), where n is the length of the password, which is polynomial time and so is a tractable problem.  Guessing the password is intractable as the only method that can be used is brute-force (try all possible password. Each extra bit added to the password doubles the number of potential passwords meaning that guessing the password has time complexity O(2n), where n is the length of the password, which is exponential time and so is an intractable problem. |

Chapter 4.4.6 Computable and non-computable problems

|  |  |
| --- | --- |
| 1a) | It is a contradiction |
| 1b) | Undecidable |
| 2ai) | No |
| 2aii) | No |
| 2b) | Yes, it terminates if x is any power of 2. |

Chapter 4.4.7 Halting problem

|  |  |
| --- | --- |
| 1) | Yes. The purpose of a compiler is to compile source code to native code, i.e. code that can be executed because a machine exists that understands the operations that the code denotes. We are excluding from this argument compilers that are designed to translate high level programs to an intermediate language. Therefore, any valid high level program can be translated into machine code provided that it is syntactically correct and written in the programming language the compiler is designed to translate. The source code form of compiler C is being processed by the executable form of compiler C when it is translated into machine code, hence the reference to treating input to the compiler C as data. |
| 2) | The Halting Problem decider cannot **decide in general** if a program will halt on particular input. In general means that the program under test is chosen from an **arbitrary set of programs**, i.e. the set of programs is constructed/chosen from the infinity of all programs. However, if the set is finite and contains all the ones that halt then the decider can decide that these will halt.  If we test compiler C on its source code and it halts then the halting problem decider will also indicate that this compiler will halt on the source code form of this compiler.  Questions 1 and 2 are in part about the fact that it is possible to decide that certain programs will halt on certain inputs but that it is not possible to write a decider that in general can decide. |
| 3a) | There are an infinite number of possible programs. |
| 3b) | By having a “dictionary” of known viruses that the program makes use of. If the dictionary is updated regularly as new viruses are discovered then this software can be effective. |
| 4) | Because they are claiming to have solved the Halting problem which has been proved to be unsolvable. |

4.5 A model of computation

Chapter 4.5.1 Turing machine

|  |  |
| --- | --- |
| 1) | …□11111□… |
| 2a) |  |
| 2b) |  |
| 3) | 011□□□□ State s  □11□□□□ State 0  □11□□□□ State 0  □11□□□□ State 0  □11□□□□ State t0  □11□□□□ State n |
| 4) | 101□□□□ State s  □01□□□□ State 1  □01□□□□ State 1  □01□□□□ State 1  □01□□□□ State t1  □0□□□□□ State r  □0□□□□□ State r  □0□□□□□ State s  □□□□□□□ State 0  □□□□□□□ State t0  □□□□□□□ State y |
| 5) | A Turing machine consists of a finite set of states, a finite alphabet, an infinite tape, a read-write head and a set of transition rules. |
| 6) | A universal machine is a machine capable of simulating any other machine. |
| 7) | A universal Turing machine (UTM) us an interpreter that reads the description of any Turing machine and the input provided to that Turing machine and faithfully executes the given Turing machine on the given input. |
| 8) | An interpreter works its way through a set of instructions identifying the next instruction then executing it. A UTM does the same – working through the instructions needed to execute a given Turing machine on the input provided. |
| 9) | A program’s instructions can be represented in the same form as data. When represented this way the program can be used as an input (data) for another program. |