Higher Computing Science
Software Design and Development - Programming
Summary Notes

Design notations

A design notation is the method we use to write down our program design.

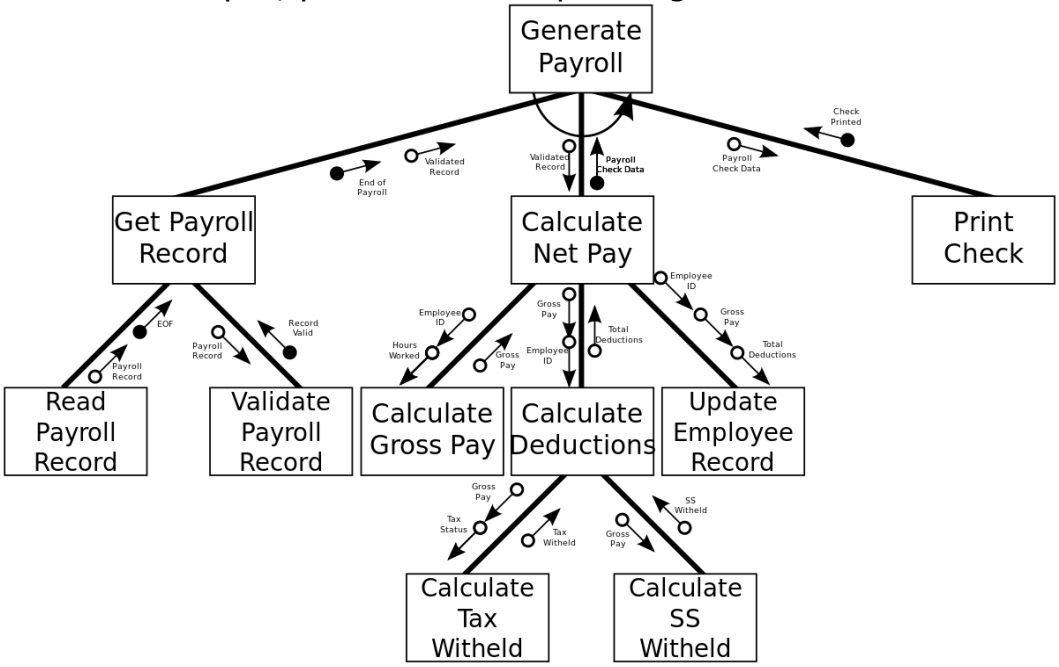
- **Pseudocode** is written using English words and is easily changed, line by line, into the chosen programming language (see all of the examples in this booklet).

  Since it is not a formal language there is not strict rules on writing pseudocode, other than it should follow a similar structure to a programming language.

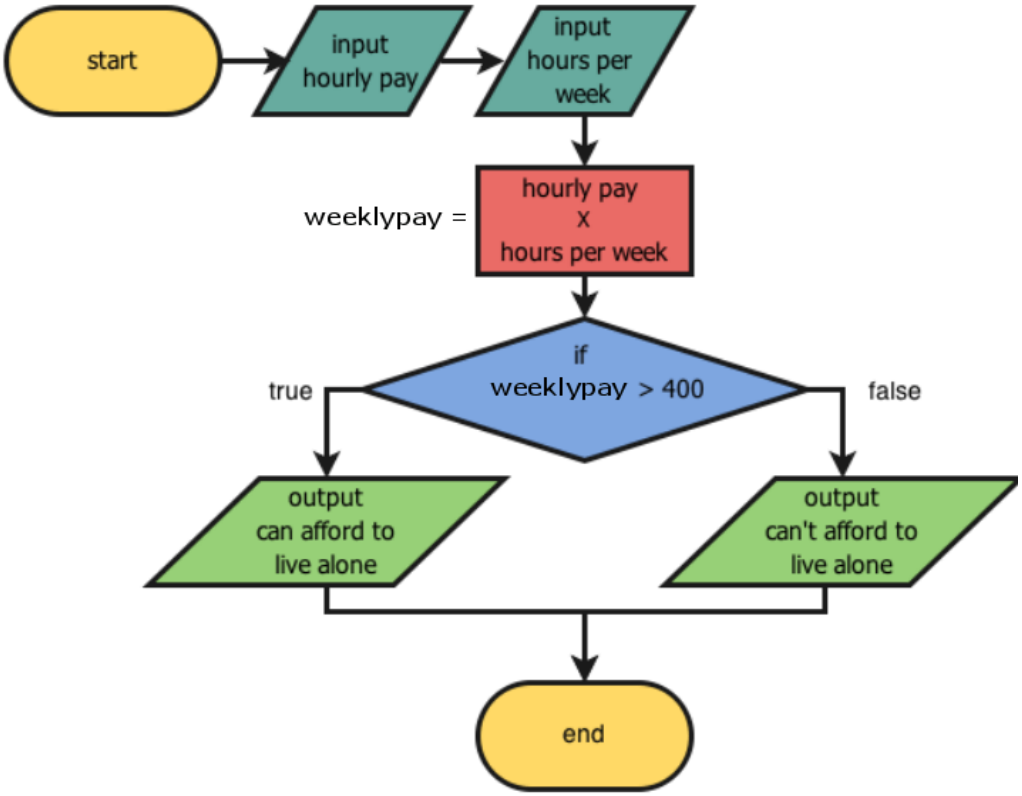
PSEUDOCODE

```
set total to zero

get list of numbers

loop through each number in the list
   add each number to total
end loop

if number more than zero
   print "it's positive" message
else
   print "it's zero or less" message
end if
```

- A **structure diagram** gives a visual representation of how the program is broken down into smaller problems. It is read from top to bottom, left to right and often represents the main input, process and output stages.



- A **flowchart** gives a visual representation of the sequence of processes in the program. It also shows the program flow, control constructs and branches in the program.



- **Wireframes** are used to design the user interface for a program. The diagram shows the layout of elements and is often annotated with details of the styling to be used.
-

Data structures and types

Two data structures are available for storing information:

- a **variable** is used to store a single item of data
- an **array** (1D array) is used to store a list of items that share the **same** data type.

Some data types which can be used are:

- **string** (for text, e.g. "Hello World")
- **character** (for a single character, e.g. "@")
- **integer** (for whole numbers, e.g. 1,2,3)
- **real** (for non-whole numbers, e.g. 3.14, 7.5, 42.12)
- **Boolean** (for True/False results)
- **records** (aka User Defined Types) can hold a variety of data types in multiple variables or arrays. Records can contain values of more than one data type.

VB examples	Description
Dim name As String	A string variable called name
Dim age As Integer	An integer variable called age
Dim names (1 To 30) As String	A string array called names, indexed 1 to 30
Dim prices (1 To 80) As Single	An array of real numbers called prices, indexed 1 to 80

Private Type record
  OrderID As Integer
  ItemName As String * 20
  InStock As Boolean
End Type
	A User Defined Type (UDT) called record, with 3 fields (integer, 20 character string, Boolean)

Dim items (1 to 50) As record	An array of records, called items, indexed 1 to 50

Scope of variables

The scope of a variable is the part of the program for which it is valid.

- **Global** variables have the whole program as their scope.
- The scope of a **local** variable is the subprogram in which it is declared.

In VB, global variables are declared in the General section at the start of the code.

Modularity

Splitting a program up into subprograms and modules aids readability and makes it easier for several programmers to work on a project at the same time, speeding up the development process.

- A **procedure** is a subprogram which carries out a particular task.
- A **function** is a subprogram which returns a single piece of data.

Procedures and functions may be defined as either:

- **Private** - can only be used in the module which contains it
- **Public** - can be used by any module

A **module library** is a collection of pre-written, pre-tested modules which is used to save time in the development process. It can also allow programmers to carry out tasks beyond their expertise.

Parameter passing

Procedures and functions may require certain pieces of information in order to do their job - these are called **parameters** (aka **formal parameters**).

Using parameter passing correctly aids modularity.

Procedure parameters may be required **by reference** or **by value**.

- **By reference** – a pointer to the location of the data is passed, so any changes to it will change the data for the rest of the program.
- **By value** – a copy of the data is passed, so any changes to it will not affect the rest of the program.

The part of the program which calls the subprogram must pass the appropriate pieces of data - these are the **arguments** (aka **actual parameters**).

VB examples	Description
Private Sub initialise ()	Defines a private procedure called initialise with no parameters.
Public Sub open_file (ByVal filename As String)	Defines a public procedure called open_file which requires a string parameter by value.
Private Function circle_area (ByVal radius As Single) As Single	Defines a private function called circle_area which requires a real parameter passed by value and returns a real result.

Pre-defined functions

Pre-defined functions are built in to the programming environment and perform useful calculations. Some predefined functions include: Int, Rnd, Sin, Cos, Len, Left.

VB examples	Description
Rnd	Returns a random real number between 0 and 0.099999
Int (single)	Returns the whole number part of a real number
Len (string)	Returns number of characters in a string
Left (string, integer)	Returns a substring taken from the left of a string
Mid (string, integer, integer)	Returns a substring taken from any point in a string

Assigning values to variables

This means "putting data into a variable".

Pseudocode examples	VB examples
SET age TO 21	age = 21
SET name TO "Sherlock"	name = "Sherlock"

Arithmetic operations

Arithmetic operations include +, -, *, /, ^ (to the power of) and mod (the remainder of a division).

Pseudocode examples	VB examples
SET answer TO 3 + 4 ^ 2 SET remainder to 17 mod 5	answer = 3 + 4 ^ 2 remainder = 17 mod 5

Concatenation

Concatenation is the process of joining strings, variables and arrays together.

Pseudocode examples	VB examples
SEND "Hello " & name TO DISPLAY	MsgBox ("Hello " & name)

Conditional statements

Conditional statements use the IF…THEN…ELSE structure.
They are used to select particular lines of code to be carried out.

Pseudocode examples	VB examples

IF age >= 17 THEN
SEND "You can learn to drive" TO DISPLAY
ELSE
SEND "You are not qualified" TO DISPLAY
END IF
	If age >= 17 Then
	MsgBox ("You can learn to drive")
	Else
	MsgBox ("You are not qualified")
	End If

IF name ≠"Moriarty" THEN
SEND "Welcome" TO DISPLAY
END IF
	If name <> "Moriarty" Then
	MsgBox ("Welcome")
	End If

Logical operators

Logical operators – AND, OR, NOT – can be used to create complex conditions.

Pseudocode examples	VB examples
IF score>100 AND score<500 THEN	If score>100 And score<500 Then
WHILE age<21 OR name<>"Watson" DO	Do While age<21 Or name<>"Watson"

Fixed loops

A fixed loop repeats a section of code a set number of times.

Pseudocode examples	VB examples
REPEAT 10 TIMES SEND name TO DISPLAY END REPEAT	For counter = 1 To 10 List1.AddItem name Next counter
FOR loop FROM 1 TO 20 DO SEND loop TO DISPLAY	For loop = 1 T0 20 List1.AddItem loop

END FOR	Next loop

Conditional loops

A conditional loop repeats a section of code either WHILE a condition is met or UNTIL a condition is met.

Pseudocode examples	VB examples
REPEAT RECEIVE response FROM KEYBOARD UNTIL response = "No"	Do response = InputBox ("Continue?") Loop Until response = "No"

WHILE response ≠ "No" DO
RECEIVE response FROM KEYBOARD
END WHILE
	Do While response <> "No"
	response = InputBox ("Continue?")
	Loop

File handling operations

It is often useful for programs to work with external data files, there are a few basic operations which can be carried out.

- **Open/Create** – a file must be opened or created before it can be read from or written to.
- **Read** – reads data from a file into a variable or array.
- **Write** – writes data into a file
- **Close** – a file must be closed once it has been used, this frees up the memory it was in.

Sequential files in Visual Basic

Sequential files are the most straightforward type, used to store a simple text file. These examples give the syntax for basic sequential file operations.

Creating/writing to a sequential file

If the file already exists, any existing data will be overwritten.
If the file doesn't exist, it will be created.

Pseudocode example	VB example
OPEN "H:\scores.txt" SEND score TO "H:\scores.txt" CLOSE "H:\scores.txt"	Open "H:\scores.txt" For Output As #1 Print #1, score Close #1

Reading from a sequential file

This will read the entire contents of the file.

Pseudocode example	VB example
OPEN "H:\scores.txt" RECEIVE data FROM "H:\scores.txt" CLOSE "H:\scores.txt"	Open "H:\scores.txt" For Input As #1 data = Input(LOF(1), 1) Close #1

Adding to a sequential file without overwriting

This will add data to the end of the file.

Pseudocode example	VB example
OPEN "H:\scores.txt"	Open "H:\scores.txt" For Append As #1

SEND score TO "H:\scores.txt" CLOSE "H:\scores.txt"	Print #1, score Close #1

Random files in Visual Basic

Random files used to store data in an organised structure using records.
By using records and fields, random files can give much more control.
For example, a specific record can be read, amended, or written to.

Creating/writing to a random file

This will add one record at a specified position.

Pseudocode example	VB example
OPEN "H:\s.txt" SEND record TO "H:\s.txt" CLOSE "H:\s.txt"	Open "H:\s.txt" For Random As #1 Len = Len(record) Put #1, recordnumber, record Close #1

Reading a single record from a random file

This will read a specific record.

Pseudocode example	VB example
OPEN "H:\s.txt" RECEIVE record FROM "H:\s.txt" CLOSE "H:\s.txt"	Open "H:\s.txt" For Random As #1 Len = Len(record) Get #1, recordnumber, record Close #1

Reading all the records from a random file

This will loop through all records in the file and read them into an array.

Pseudocode example	VB example
OPEN "H:\s.txt" FOR EACH record FROM "H:\s.txt" DO RECEIVE record FROM "H:\s.txt" END FOR EACH CLOSE "H:\s.txt"	Open "H:\s.txt" For Random As #1 Len = Len(record) numberofrecords = LOF(1)/len(record) For counter = 1 to numberofrecords Get #1, counter, record Next counter Close #1

Errors

There are 3 main types of programming error.

- **Syntax** – incorrect use of the programming language.
  e.g. Typing *Nxt counter* instead of *Next counter*
- **Execution** – errors while the program is running, usually causing an error message.
  e.g. *Type mismatch* error if the program tries to store text in an integer variable.
- **Logic** – no syntax or execution error but program doesn't produce correct results.
  e.g. The program should only allow input of numbers from 1 to 10, but it allows any number to be input.

Debugging

Various debugging techniques and tools are available to programmers:

- **Dry runs** - Working through the listing using pencil and paper.
- **Trace tables** - Used to note variable values when carrying out a dry run.

- **Trace tools** - Allow access to extra information which is normally hidden, such as the call stack (the list of active subprograms).
- **Breakpoints** - A marker in the code where the program execution is to be paused. Allows the programmer to pinpoint the moment where an error is occurring.
- **Watchpoints** -similar to a breakpoint but used to monitor the value stored in a variable or check a certain condition. The program will stop when the specified criteria is met and details of the current state can be displayed.

Testing

Software is tested methodically, following a systematic **test plan**, to make sure it is free from errors. A good test plan should include the following types of test data:

- **Normal** – acceptable test data, well within acceptable limits.
- **Extreme** – acceptable test data but on the limits of what is acceptable.
- **Exceptional** – unacceptable test data, outwith acceptable limits.

Including these types of test data will help to ensure a program is tested **comprehensively**.

Example
A program asks the user to input a number from 1 to 10. Normal test data – 4, 5, 6 Extreme test data – 1, 10 Exceptional test data – 233, -15, A, %

Readability

How easily the program can be understood by another programmer.
Readability can be improved by adding:

- **internal commentary** – comments in the program code to explain what it is doing.
- **meaningful identifiers** – using sensible names for variables, arrays and subprograms, e.g. *score* rather than *s*.
- **indentation** – using the tab key to help show where control structures start and finish.
- **white space (blank lines)** – to help separate sections of code.

Standard algorithms – Linear search

- A standard algorithm used to check if a value is in a list.
- Loops through every item in list and compares it to the target item, setting a flag variable to true if they match.
- Can be made more efficient by only continuing the search if the target item hasn't been found yet.

Basic algorithm	**More efficient algorithm**

RECEIVE target FROM KEYBOARD
FOR EACH item FROM list DO
  IF item = target THEN
  found = TRUE
  END IF
END FOR EACH
 	
IF found = true THEN
	RECEIVE target FROM KEYBOARD
	SET found TO false
	WHILE found = false AND not end of list DO
	  IF current_item in list = target THEN
	  found = TRUE
	  END IF
	END WHILE

```
    SEND "Target found" TO DISPLAY
ELSE
    SEND "Target not found" TO DISPLAY
END IF
```

```
IF found = true THEN
    SEND "Target found" TO DISPLAY
ELSE
    SEND "Target not found" TO DISPLAY
END IF
```

Standard algorithms – Count occurrences

- A standard algorithm used to check how many times a value appears in a list.
- Loops through every item in list and compares it to the target item, adding 1 to a counter if they match.

Count occurrences

```
SET hits TO 0
RECEIVE target FROM KEYBOARD
FOR EACH item FROM list DO
    IF item = target THEN
    SET hits TO hits + 1
    END IF
END FOR EACH
SEND hits TO DISPLAY
```

Standard algorithms – Find maximum / find minimum

- Standard algorithms used to identify the highest / lowest values in a list.
- Loops through every item in list and compares it to the target item, assigning a new value to maximum/minimum if necessary

Finding maximum

```
SET maximum TO list[0]
FOR EACH item FROM list DO
    IF item > maximum THEN
    SET maximum TO item
    END IF
END FOR EACH
SEND maximum TO DISPLAY
```

Finding minimum

```
SET minimum TO list[0]
FOR EACH item FROM list DO
    IF item < minimum THEN
    SET minimum TO item
    END IF
END FOR EACH
SEND minimum TO DISPLAY
```

Languages and environments

Low-level languages

- Low level languages are machine dependent and require a detailed understanding of the specific computer's processor, registers, memory layout, etc.
- Machine code is the lowest level language.
- Slightly higher level is assembly language, which uses mnemonics like LD, RET, etc to manipulate data between the processor and memory.
- The purpose of low level programming is to create highly efficient programs, optimised for processor and memory usage.
- Low level programs are extremely difficult to program/debug as they contain no high level constructs (repetition, selection, etc).

Machine code

- Originally the only means of programming a computer (in the 1950s).
- Programs would only run on the machine architecture on which they were created.
- Very difficult to read and maintain.
- Difficult and time-consuming to remember or look up opcodes.

Assembly language

- Addressed the problem of difficult to remember opcodes, by replacing them with mnemonics (e.g. JMP instead of 1001).
- Much easier to write, read and maintain than machine code.
- Required an assembler to translate programs into machine code.

High-level languages

- Addressed the need for more powerful programming languages.
- Code resembles English and is much easier to read and maintain.
- One line of HLL code would take approximately 10 to 20 lines of low level code to carry out the same action.
- Gives access to control structures such as repetition and selection.
- Gives access to data structures such as arrays.
- Requires a **compiler** to translate programs into machine code versions.

Procedural languages

- A procedural language is a HLL in which the user sets out a list of instructions in the correct sequence in order to solve a problem.
- It has a definite start and finish point and the instructions are followed in sequence.

Declarative languages

- Rather than instructions, facts and rules are entered (this is called the knowledge base), and the language uses these to find solutions to queries which are entered.
- Declarative languages are typically used in the development of Artificial Intelligence and expert systems.

Object-oriented languages

- Object oriented programming is a different approach to program design.
- Use of object-oriented techniques can reduce coding and development time.
- Objects define the data (aka properties, a description of the object) and the methods (aka operations, or things that the object can do) that can be used to manipulate the data. Every **instance** of the object will have the same properties and methods.

Development process

The software development process follows these stages:

- **Analysis** – clarifying requirements and producing the **software specification** (precise details of the requirements, also a legally binding contract between client and developer). **Systems analyst** interviews and observes clients.

- **Design** – planning the software, based on the specification. Creating wireframes for user interface and using **top-down design** to produce an algorithm.

- **Implementation** – Choosing a suitable programming language (considering available data types, control structures, operating system, expertise, etc) and creating the program based on the design.

- **Testing** – using a systematic test plan to make sure the program is free from errors. An **independent test group** may be used to provide more rigorous, unbiased testing.

- **Documentation** – writing formal documentation such as the EULA (copyright restrictions), technical guide (installation and hardware requirements) and user guide (how to use the software).

- **Evaluation** - judging the program in relation to various criteria – robustness, maintainability, efficiency, portability, reliability, fitness for purpose, etc.

- **Maintenance** – making changes to correct, improve or adapt the program.
  Corrective maintenance involves fixing errors which weren't previously spotted.
  Perfective maintenance involves improving the software by adding new features or streamlining code.
  Adaptive maintenance changes the software to run in a new environment, such as a new OS or different hardware.

The development process is **iterative**, in that earlier stages may have to be repeated as a result of new information.

Rapid application development (RAD)

A development approach which involves quickly building a small-scale prototype then repeatedly evaluating and improving until the final product is achieved. Design time is reduced and problems are caught early in the process.

Agile programming

A development approach which focuses on flexibility and communication, recognising that the analysis of a large-scale project is very difficult. The client is consulted regularly and parts of the solution are delivered and evaluated upon their completion.

Contemporary developments

- **Software development environments** have become more graphical and include many debugging tools to help the programmer.
- Improvements in hardware have allowed more complex AI techniques to be used to create ever more advanced **intelligent systems** such as robots.
- The use of **online systems** has increased as network bandwidths, processor power and storage capacities have increased.