***** These answers are for the older version of these review questions, so some are incorrect *****
***** especially the algorithms which are written in PURE instead of Java *****

QUESTIONS: Airplane Reservations

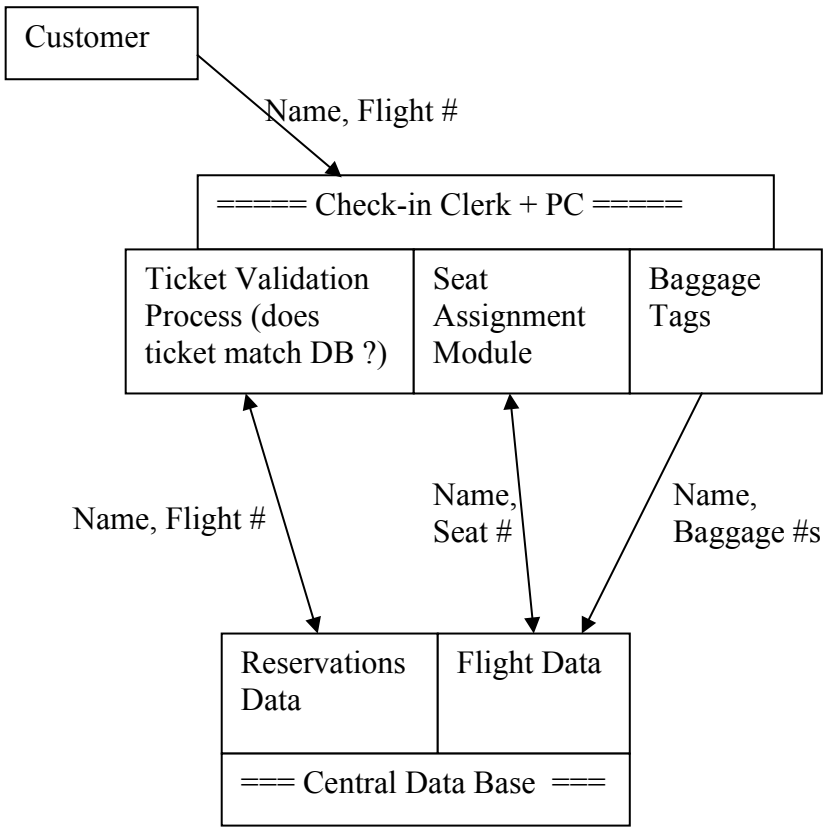
- Describe three significantly different **types of data** in this system.
  * Customer names, flight numbers, prices
- Describe three significantly different **users** of this system.
  * Travel agent, customer, airline companies
- For one of the **types of data**, estimate the **total quantity of input** required per week.
  * Customer names : if the airline has 20 flights per day, each with 100 customers, that would be 20*100*7 = 14000 names per week. If the name is in a 30 character field, that is 420 KB per week
- Describe two uses of a **computer terminal** by a travel agent in this system.
  * (1) Getting flight information about times, dates, and prices
  * (2) reserving a ticket for a specific customer on a specific flight
- Tickets can be printed at the travel agency. What **output device** is required?
  * printed output onto a ticket, which is a specially formatted form

  Does it have any special requirements other than normal functions?
  * The data must be printed into the boxes on the preprinted ticket.
- Describe a **processing feature** in this system which is **automated** and could probably not be done **manually.**
  * booking reservations could be done manually
- If a customer is to be **reminded automatically** of an upcoming flight, describe the **processing** required for this reminder, including **accessing stored data** and **producing output.**
  * We could write a rough algorithm for this

```
search through all future reservations
find all reservations for the date 1 week from today
for each reservation
  get the corresponding name and phone/addreess/email (choose)
  notify that person
next reservation
```

```
It may be useful to identify "family" reservations, and only send a
single reminder to the whole family.  This could be done be denoting
a customer as "notifiable" or "not-notifiable"
```

- A **flight attendant** needs to **input** data into this system – what data? Under what circumstances?
  * When the plane is boarding, the flight attendant must check that all the correct customers
  * board the plane. This could be done with a printed list of reservations, then checking off
  * whether each seat is full. It could also be done in a portable computer. The portable computer
  * has the advantage that it is easier and quicker to send this data back into the main system.
  * The data must be compared to the actual reservations very quickly, because take-off will
  * be delayed if it does not match.
- Explain what a **data-flow-diagram** is, and draw one for the process of **checking-in** at the airport.
  * A data-flow diagram has a box for each input module, each output module, each storage module,
  * and each processing (e.g. sorting) module. These are connected by arrows which show what
  * data flows between the various modules. Module boxes can represent hardware or software or both.


Customer
Name, Flight #
===== Check-in Clerk + PC =====
Ticket Validation Process (does ticket match DB ?)
Seat Assignment Module
Baggage Tags
Name, Flight #
Name, Seat #
Name, Baggage #s
Reservations Data
Flight Data
=== Central Data Base ===

STAGES in system development:
The **investigation/analysis** stage should produce a **feasibility study** which involves:

- **ANALYSTS (and USERS)**
- User's goals and requests
- Stories from users, describing problems and needs
- Sample data
- Possible hardware choices (perhaps several alternatives)

-- **Explain two reasons** why it is desirable to have **different people** involved in the different stages.
* (1) The various people have different areas of expertise – it is bad if they are working outside this area,
* and would slow things down and perhaps result in poor decisions.
* (2) Some decisions in one area may be **reversed** by other people in other areas – this is good.
-- In which stage does **module-level testing** occur?
* In the programming (implementation) stage.
-- In which two stages would a **questionnaire** be usefule?
* Investigation/anslysis and Maintenance/review.

1.3 Software Design

* OMR = Optical Mark Recognition – pencil bubble sheets commonly used for multiple choice tests
* GUI = Graphical User Interface, as opposed to a Command Line (Text) interface
* (1) store it in a string as "05/23/02"
(2) store in an array, as Date[1] = 5 Date[2] = 23 Date[3] = 2002
(3) Store it in a long integer as 20020523.
* 1234567890987654321 = it's too big - overflow
* 1 e –123456789 = too small = underflow
* All the ASCII codes below 32 represent printer control codes, not characters. Zero (0) is "never" used.
* A list of names. If there are too many names, it might **overflow** the size (dimension) of the array.
* Names(1000) as string, Ages(1000) as integer
The Name of one person is stored in the Names array, and his/her age is store in the same position in Ages.
* If the file is too large, it might not fit in the RAM, so it won't fit in an array.
* To represent a chess-board, to represent seating in a theatre

A common **algorithm** is the **bubble-sort.**
* If the array is already sorted, the bubble-sort will make no swaps in the first pass, and could then stop.
* Use a **parameter** for the name of the array (e.g. LIST), then call SORT(Names), SORT(Cities), ...
* It has nested loops, so 1000 items need 1000*1000 iterations = 1 million
8000 items needs 8000*8000 = 64 million iterations, so it takes 64 times longer = 640 seconds = 10.7 min
* A selection sort will be faster on scrambled up data, because each pass is shorter than the previous one.
* If the file is already sorted, the bubble sort **could** quit after one pass, whereas a selection sort always requires the same number of iterations – it cannot stop early.

Another common algorithm is a **sequential search.**
* If it finds the name it is looking for, or in a sorted list if the target name is smaller than the current item.
* Faster in an array because disk-access is much slower than memory access
* Binary search – start in the middle .....etc
* Takes 8 times longer (single loop), so 80 sec.

1.4 Constructs

* scope = where in the program can the variable be used
* global variables have larger scope
* a parameter is very similar to a local variable
* A **record** like:

```
class Student
{  String name;
   int age;
   String[] address = new String[4];
}
```

* Something like (x>4) , (name = "Bob"), using the operations > , < , =
  The result of a comparison is a boolean value **true/false**
* Something like (x < 5) and not eof(datafile), using boolean operations AND, OR, NOT, XOR
  The result of a boolean expression is a boolean value **true/false**
* In PURE, the **operator precedence** is:

  1 : brackets
  2 : arithmetic operators, * / before + -
  3 : comparisons, < > = # in order from left to right
  4 : boolean : not first, then and, then or/xor

* A **record** can contain many different types of data in different ___fields_____ .
* a **selection construct = if..then..**
* **looping constructs** : **for..enfor while..endwhile repeat..until..**
* **iteration construct** : the same as a **looping construct**
* **sentinel values** are strange data values like "XXX" which mark the end of an array (or a file)
* **flag :** a **boolean** variable to keep track of whether something has happened or not,
  for example FOUND in a search algorithm.
* **by-reference :** any changes to the parameter inside the procedure are copied back to the calling routine
  by-value: input only – changes are not copied back to the calling routine.

Long Question

```
function COUNT(val LIST string array)
   result integer
   declare C,POS integer
   POS <-- 1
   C <-- 0
   while ( LIST[POS] <> "XXX" ) do
      C <-- C + 1
      POS <-- POS + 1
   endwhile
   return C
endfunction

procedure REVERSE(val LIST string array)
   declare X, START integer
   START <-- COUNT(LIST)-1
   for X <-- START downto 1 do
      output LIST[X]
   endfor
endprocedure
```

1.4 Errors and Testing

```
function ADDUP(val N integer)
   result integer
   declare X, SUM integer
   SUM <-- 0
   for X <-- 1 to N
      SUM <-- SUM + X
   endfor
   return SUM
endfunction
```

* incorrect answer = logic error
* missing letter = syntax error
* no, the result would be zero, no problem
* The compiler detects a syntax error when something is written incorrectly.
  But run-timer errors result from bad data or some other situation which the compiler cannot predict.
* ADDUP(10) , expected answer = 55
  ADDUP(0) , expected answer 0
  ADDUP(1), expected answer 1
* A very large value 123465691261928034619023 might cause an overflow.
  prevent this with an if..then..:

```
If N < 1000000 then
    do the procedure
  else
    output error message
    return -1
endif
```

* Instant error messages when a syntax error is typed
  DEBUG (trace) mode showing which statements are being executed
  Syntax highlighting (colors)
  Break at run-time error showing offending statement
* Print the program listing on paper, take a pencil, and READ the program,
  trying to "run" the program step by step with sample data
* **infinite loop** = run-time error **and** logic error

Longer Question

Consider the following algorithm:

```
main
   output ROUNDED(19.9559);
endmain

function ROUNDING(val NUM integer)
   result integer
   return truncate(NUM + 0.5)
endfunction
```

* syntax error : ROUNDED and ROUNDING
* type mismatch caused by parameters : val NUM real would be better.
* output = 20
* wrong output = Logic error

1.5 Documentation

* Internal docs talk about program code and data structures (how the program was written)
  while user docs talk about how to USE the program
  Internal docs usually appear inside the program listing
  while user docs are contained in a separate book
  Internal docs are used when the program needs to be fixed or changed
  while user docs talk about how to install and maintain the program
* A **technical writer** should write them. This person should have some technical understanding
  and significant level of understanding of the application area (user perspective).
  This person serves as an interface between the programmers (developers) and the users.

NEW Language

* the **position** of the **remote** identifier is a **syntax** issue
* the choice of the word **remote** or **extern** is a **semantics** issue
* **$ sign** is both syntax and semantics issue
* Interpretter must translate each command over and over again,
  and will be making long-distance connections over and over again, which
  might be worse than with a compiler
* intended for both LAN and WAN, as there will also be members of the programming team
  locally in Germany
* HTML is **not** a programming language
* The CASE Tool should run on the client, because it must have access to the Indian programmer's
  part of the project, which would not work from the server side.

==========

2.2 Architecture

They are now matched here:

Device	Application
OCR	change a printed page back into a word-processing document
sensor	respond to movement or light signals
printer	create bar-coded sticky labels for library books
MICR	print an ID number on a bank chequie
mouse	select from a drop-down menu on a PC
scanner	copy a photograph into the computer
modem	transmit packets of data from one computer to another
graphics tablet	draw accurate pictures with a stylus
cell phone	convert analog sound to digital data and transmit it from various locations
LCD panel	display data in a portable computer
keyboard	give commands in a command-line interface
digital camera	capture pictures which can be displayed over the Internet
speech recognition	give commands to a computer when your hands are busy
sound card	input music
plotter	draw accurate architectural diagrams, without jaggies
touch-screen	start a program in a pocket-organizer

2.2.2 Chips

The ALU and CU are two parts of the ___CPU_____. The ALU performs _arithmetic____ and

___logic_____ operations. The ___CU_____ controls communications across the data bus.

The data bus contains 32 (or maybe 64) __parallel_ circuits,

each of which can carry one ____bit_____ of data.

The CPU uses the data bus to fetch and store data in the _____RAM_____.

But to speed up operations, the data bus does not connect directly between the CPU and the RAM.

The CPU uses the _____cache_____ for temporary storage, and **it** is connected to the RAM.

The speed of the CPU might be 1 ___GIGA_____ Herz. This means __1 billion_____ cycles per second.

If the data bus is 64 ___bits__ wide, and runs at 266 ___megaherz__,

then it can transfer data at a speed of _8 bytes * 266 MegaHz = _2128__ MegaBytes / Sec.

This is the same as __2.1____ GigaBytes per second.

Earlier computers did not run at such high speeds. Running a CPU at a higher speed generates a lot of __heat__.

Cooling fans can only help a certain amount. Super computers use liquid nitrogen to cool the circuits and chips,

which can then run at very high frequencies without damage. In PCs, the solution is to make the circuits in the

chips __smaller____, reducing the size from 0.25 microns to 0.13 microns, and then also reducing the

voltage from 3 V to 1.5 V. Using less energy produces less heat, so the chip can run at a faster frequency

without overheating. It is also possible to place more ___transistors_ on the chip in the same space.

A Pentium chip contains over 25 ___million____ transistors.

In RAM, a transistor represents one __bit____. 8 __bits___ makes 1 ___byte____

Modern PCs contain 128 ___megabytes___ of RAM, or more. A hard-disk might store 40 ___GigaBytes_ of

data. The hard-disk can store _40 * 1024 / 1.44 = 28,444 _ times as much data as the RAM.
When a program runs, it must
be copied into the ___RAM_____. If several programs are running at once (__Multitasking_____),

the RAM could get full. Then the CPU must **swap** some of the memory out onto the ____hard disk____,

to free up more memory for other programs. This process is called ___virtual_____ memory, because the

computer seems to have more memory than the actual physical RAM.

2.2 Storage

T = Tera = 2 ^ __40____ 1 Terabyte is equal to __1 million_ MegaBytes.

G = Giga = 2^ ____30__ A 40 GigaByte hard disk can hold as much as _28,444__ floppy diskettes.

M = Mega = 2^ __20____ Using a 56 K modem, the fastest possible download of 1 MB is _142___ sec.

K = Kilo = 2^___10____ A hard-disk sector containing 512 bytes is the same as __0.5 __ KB.

A 32 bit address bus can carry addresses between 0 and ____4 billion__ and address a maximum

of _____4 GigaBytes_____ RAM.

A _____serial_____ interface has only 1 data wire, and transmits 1 bit after the next.

If it runs at 240 KHz, it can trasmit _240 K / 8 = 30 KB per second.

A __parallel____ interface uses 8 data wires, and thus can transmit an entire byte at one time.

==========

* Kilometers are a base-10 measurement, and 1000 is a power of 10 (10^3).
  KiloBytes are counted in binary, and 1024 is a power of 2 (2^10)
* In sequential access, the data must all be read in order, from the beginning to the end.
  In direct access, it is possible to "jump" to any position in the file without reading the previous data.
* Backups are commonly done on tape-drives, which can only be used in sequential access mode
  Tape is used because it is cheap and can be removed and stored somewhere else for safety,
  and the speed of the backup process is not an important issue.
* Smaller circuits squeeze more circuit elements into the same space. The manufacturing cost
  depends more on the physical size of the chip – if that stays the same, the cost stays the same.
* The **pits (holes)** on the DVD disk are smaller and packed closer together than the CD-ROM

==========

* to "jump" into the middle of the file, the computer must **calculate** the number of bytes between the beginning of the file and the desired record. This is only possible if all records are the same size (in bytes)

==========

Long Question

A new **pocket PC** contains no disk drive. Instead it uses **flash memory**. This is a large amount of **RAM** (e.g. 64 MB) with a constant power supply, so the data never gets erased. A special design achieves this without using very much power, so batteries can last several weeks or months. But flash memory is expensive.

* The OS seldom changes, but is used very often, so the ROM is an efficient solution.
* Link cable to a PC, copy data to PC disk drive
* Screen blanking saves power – an important issue in portable computers
* The screen is far too small to display web-screens – typically something like 300x200 pixels.
* There is no keyboard, so speech-recognition is a very attractive possibility for data input.

2.3 Systems

* Windows is a multi-tasking GUI.
* The server does not need a GUI, but requires very high reliability and very efficient multi-tasking
* Multi-tasking
* A main-frame probably is a mult-user system, so multi-processors is a great advantage.
* Data Files, printers, disk storage space
* E-mail client (your PC) and the E-mail server (collects and transmits mail)
* Batch proceed through the entire process without user interaction
* Posting transactions (adding/subtracting money) to user account balances
* Verifying a PIN code for an ATM transaction
* Searching for overdue books and printing warning notices (done daily or weekly)
* Checking out a book
* Anything that happens in a library is not urgent – it can wait
* Air-traffic control **cannot** wait – it must give up-to-date information constantly
* intensive care = real-time

2.4 Data Representation

Analog/Digital

Which of the following involve DAC or ADC? Which do not?

- Storing data on a disk drive **Digital**
- Playing a phonograph record **Analog**
- Using a phone-card in a .... **Digital**
- Printing from a PC to a laser printer **Digital**
- Photography with a digital camera **ADC**
- Playing CD music on a PC speaker **DAC**
- Measuring temperatures and storing them **ADC**
- Controlling an industrial robot **DAC**
- Audio sampling (what's that?) **ADC**
- Scanning and OCR **ADC**

Binary

* 24-bit color is called **true color.** How many different colors are there? 2^24 = 16 million
* One byte can contain any positive number between _0_____ and __255__.
* How many bits are needed to store the number 1,000,000 ? **20 bits**
* Write the following IP address as a 32-bit binary number: 179.123.10.99
  The four numbers are individual bytes, and can be converted independentley:
  10110011 01111011 00001010 01100011

* If the byte 10101100 is transmitted, what **parity bit** should be transmitted with it (even parity)? **0**
* Explain what a **checksum** is. Is it most appropriate with a byte, a packet, a file, or a disk drive?
  Add up all the bytes, and send along the sum with the transmission.
  Most appropriate for a packet.

****** These answers are for the older version of the review questions - some are not correct *****
****** Especially the algorithm answers, which are written in PURE *****

4.1 Numbers

Short questions
==========
* 2 ^ 24 – 1 = 16777215 (24 bit color = true color = 16 million)
* 10101100 in twos comp = –128+32+8+4 = -84
* **unsigned** 10101100 = +128+32+8+4 = 172
* 999 to binary ... algorithm: mod 2, div 2, mod 2 , div 2 ..., write mod's backward

```
                999   mod  2 = 1
  999 div 2 =   499.. mod  2  = 1
  499 div 2 =   249.. mod  2  = 1
  249 div 2 =   124.. mod  2  = 0
  124 div 2 =   62  .. mod  2  = 0
   ......
  result =   0000001111100111
```

* A B 9 0 hex = convert individual digits to 4-bit binary = 1010 1011 1001 0000

```
*       10010011  = 147
        00001111  =  15
        ========
       10100010  = 162
```

* 8 = 2^3, so multiplying by 8 moves the binary point 3 places to the right (add zeros if needed)

* 3.25 dec = 2 + 1 + ¼ = 11.01 bin

* 1/10 = 0.0001100110011001100..... bin = is a repeating “becimal”
* 1234.56 = 1.23456 e 3 mantissa = 1.23456, exponent = 3
* 1.5 dec = 1.1 bin, so this only requires 2 bits
* Using 2 bits, it is only possible to write 4 different numbers: 0.0, 0.1, 1.0 , 1.1
  This is not very useful, only storing 4 different numbers.

Long Question
==========
* (a) No place for negative sign
* (b) .1111 b 0111 = ½ + ¼ + 1/8 + 1/16 * 2 ^ 7
  .1111 move binary point 7 places = 1111000 = 120
* (c) Overflow
* (d) smallest = .0001 b 1 111 = .00000000001 = 2^-11
* (e) 0.25 can be written in 5 different ways:
  0.25 dec = .0010 b 0001 = .0001 b 0010 = .1 b 1001 = .01 b 0000 = .01 b 1000
* (f) (i) .01 b 0000 = 0.25 .001 b 0000 = .125 .0011 b 0000 = .1875
  .1101 b 1010 = 1/8 + 1/16 + 1/64 = 0.203125 this is best best you can do
  The "correct" value is: .001100 1100110011001100.. which repeats infinitely
* (ii) truncation error
* (g) underflow = 2 ^ -12 or smaller
* (h) save space ?? Where it is this needed?
  Say in a pocket computer, or a cell phone, where real numbers are not very useful
  Or in a device requiring communication but can’t have a good interface
(i) myIQ = “0.2”
  disadvantage: can't ADD or calculate with strings - that would need to be programmed

Short Questions

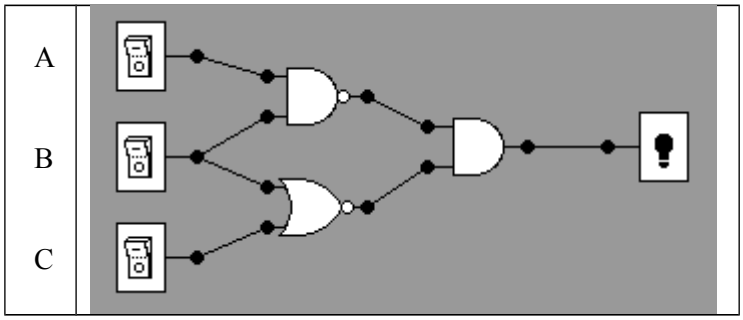
* **order of operations** (operator precedence) :
  - brackets
  - not done first
  - and, nand (from left to right)
  - or, nor, xor (from left to right)

Write a **truth table** showing all the possible values of : (p AND q) OR (p XOR q)

*

p	q	p AND q	p XOR q	(p AND q) OR (p XOR q)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

* Draw a circuit which is equivalent to : not (a AND b) AND not (b OR C)
  Notice that **not (a AND b) == a NAND b** and **not(b OR c) == b NOR c**



Draw the **simplest** circuit equivalent to this truth table – the best answer has only 2 gates.

a	**b**	**c**	**Output Value**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

* 1st try = (A and B and C) or (A and not B and C) or (not A and B and C)

  better = (A and C) or (B and C)

  best = (A or B) and C

* Write a simpler Boolean expression equivalent to : **not (not A and not B) or not C**
  De Morgan's law = not(A or B) == not a AND not B
  DeMorgan's law = not(A and B) == not A or not B
  ===> (not not A OR not not B) or (not C) === (A or B) or not C

* a truth table for a circuit with 4 inputs has **2 ^ 4 = 16 rows**

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
.....
1 1 1 1

By any appropriate means, show that these two expressions are **logically equivalent:**
(p NAND q) AND (p NOR q) == p NOR q
* Use a truth table:

p	q	A = p NAND q	B = p NOR q	A AND B	p NOR q
0	0	1	1	1	1
0	1	1	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0

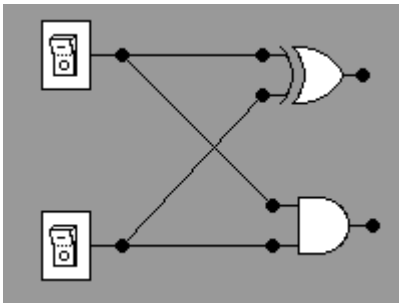
* **Half Adder –** A half-adder adds two bits (A,B) , and produces a **Sum** bit and a **Carry** bit.
The logic is in the tables below

A B S C
----------0
0 0 0
0 1 1 0
1 0 1 0
1 1 0 1

S = A xor B
C = A and B

A

B

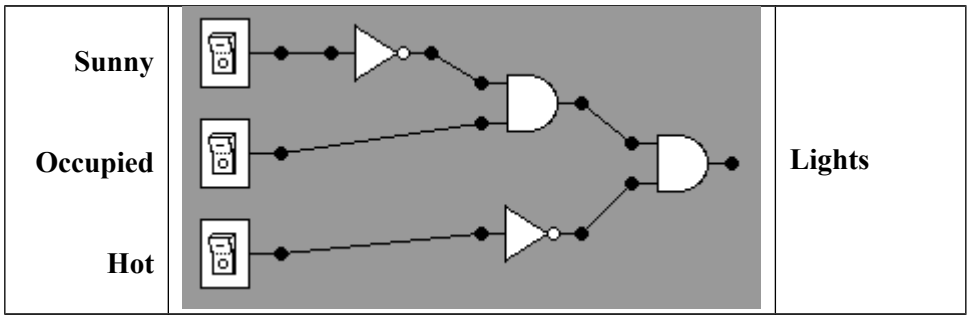


S

C

4.2 Boolean Logic : Long Question

***** (a) **not Sunny AND occupied AND not hot**

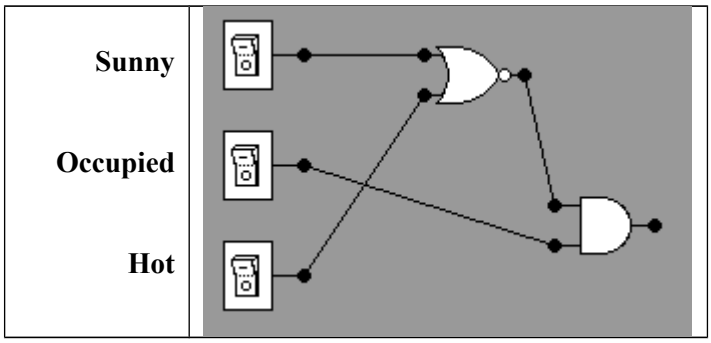


* (b) **not(** Sunny **or** Hot) **and** Occupied (2nd choice)

Sun	**Occ**	**Hot**	not Sun AND Occ AND not Hot	Sun or Hot A	not (sun or hot) B	B and Occ
0	0	0	0	0	1	0
0	0	1	0	1	0	0
0	1	0	1	0	1	1
0	1	1	0	1	0	0
1	0	0	0	1	0	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	0	1	0	0

This ========== is the same as ========== This

* (c) Use a **NOR** gate for **not (Sun OR Hot)**



* (d) **if** (Sunny = false) **and** (Occupied = true) **and** (Hot = false) **and** (Master = true)

* (e) For an **embedded system** like a traffic light, or any system requiring a large current or voltage, or any system where an entire computer is neither available nor practical, and where the logic is very simple, then a few logic gates can be used instead of an entire computer running a program. Its also cheaper and more reliable (for example, doesn't require a hard-disk).

4.3 Modulo Arithmetic (mod,div)

Short Questions
==========

* (12345 div 1000) mod 100 = 12 mod 100 = **12**
* Calculate each of the following:
  12345 mod 10 = **5** , 12345 div 10 = **1234**
  1234 mod 10 = **4** , 1234 div 10 = **123**
  123 mod 10 = **3** , 123 div 10 = **12**
  Notice that the mod operations produce the individual digits of the original number.

*

```
public int addUpDigits(int num)
{  int sum = 0;
    int digit = 0;
   while (NUM != 0)
   {  digit = num % 10;
       sum = sum + digit;
       num = num / 10;
   }
   return sum;
}
```

* Check-sum with weighting factors

```
public int addUpDigits(int num)
{  int sum = 0;
    int digit = 0;
    int weight = 3;
   while (NUM != 0)
   {  digit = num % 10;
       sum = sum + digit * weight;
       weight = weight + 2;
       num = num / 10;
   }
   return sum;
}
```

* The use of weighting factors means that an error of reversing two digits (e.g. 79 and 97) is detected. A simple sum would not detect this error.

4.3 Modulo Arihtmetic

Long Question
==========

* (a) The number of license plate codes is = 26 * 26 * 26 * 10 * 10 * 10 = 17,576,000 = 17.5 million

* (b) (65*11 + 66*13 + 67*15 + 7*17 + 8*19 + 9*21) **mod** 1000000 = 3038 mod 1000000 = 3038

* (c)

X	V	Calculate	T
	11		0
1		(65 mod 16) * 11 = 1 * 11 = 11	11
	11+1 = 12		
2		(65 mod 16) * 12 = 12	23
	12 + 2 = 14		
3		(65 mod 16) * 14 = 14	37
	14 + 3 = 17		
4		(57 mod 16) * 17 = 9 * 17 = 153	190
	17 + 4 = 21		
5		(57 mod 16) * 21 = 189	379
	21 + 5 = 26		
6		(57 mod 16) * 26 = 234	613

* (d) When the C is in a different place, it has a different **weighting factor**, so the total is different.

* (e) The ASCII code of "S" is exactly 16 more than the ASCII code of "C". The same is true for A and Q, and B and R. So the **mod 16** result is the same, and thus the same total result.

* (f) When a collision occurs, the program must store the second item in some other free cell in the array, for example the next cell or the one after that. But it must also keep track of the fact that the cell is now occupied. This could be done by storing 0 or "" in all the cells before starting the program.

* (g) The largest value for a letter must be **letter mod 16 = 15.** A larger value is not possible.
This is given by the letter "O", so "OOO999" gives the largest code, which is
15 * 11 + 15 * 12 + 15 * 14 + 9 * 17 + 9 * 21 + 9 * 26 = 1146
If this is the largest possible code, then an array with a million cells is a waste of space.
All the values are well below 1 million, so the **mod 1000000** calculation does nothing.

5.1 Terminology
Short Questions
==========
* **any of + - * / ^ .** Binary here means it has **two operands .**
* **and, or, xor**
* An **argument** is another name for a parameter
* **answer = sqrt(power(A,2) + power(B,2))**
  operands : A , 2 , B , 2
  unary operators : sqrt , power, power
  binary operator : +
* A*(B+C) / D ==> A B C + * D / also called **RPN (Reverse Polish Notation)**
* Parameters and local variables are stored in the **system stack. I**f a procedure is recursive, every time it calls itself it stores more local variables and parameters in the system stack. If it never exits, these never get erased, and eventually the stack gets full and **overflows.**
* **Pop** and **Push** are standard operations for a **STACK**
* A **QUEUE** can be either **linear** or **circular.** If it is **CIRCULAR** , overflow errors are less likely.
* Stack underflow occurs when the stack is empty, and the program attempts to **POP** a value.
* a **stack** to store print jobs in a server ==> print jobs would be processed in reverse order, so the first print job might sit around all day and never be processed.
* **Iterative = Loops (For..next, While..wend, Repeat..until)**
  Recursive = A procedure or function which calls itself.

Long Question
==========
*** (a)**

```
function CheckBrackets(val FORMULA string)
   result boolean
   declare X integer, C character, LAST character
   declare STACK character array[1..100], TOP integer
   TOP <-- 0                              /* initialize the stack */
   for X <-- 1 upto length(FORMULA) do
      C <-- copy(FORMULA,X,1)
      if (C = "(") or (C = "[") or (C = "{") then
         TOP <-- TOP + 1                  /* PUSH */
         STACK[TOP] <-- C
       elsif (C = ")") or (C = "]") or (C = "}") then
         if TOP = 0 then
            return false                  /* Stack Underflow */
         endif
         LAST <-- STACK[TOP]
         TOP <-- TOP - 1
         if (C = ")" and LAST # "(") or
            (C = "]" and LAST # "[") or
            (C = "}" and LAST # "{") then
            return false                    /* bracket types don't match
*\
         endif
      endif
   endfor
   if TOP = 0 then
       return true                  /* brackets are okay */
     else
       return false
   endif
endfunction
```

* (b) The **previous** brackets must be retrieved in reverse order.
* (c) **for...endfor repeat..until while..endwhile**
* (d) It only deals with single characters, not whole words like "for" and "endfor".

5.2 Algorithms

==========

Array Algorithms

In these answers, corrections are in ***BOLD ITALICS.*** Deletions are ~~struck through~~

==========

```
ADD a new item to the end of the array (append)
SIZE parameter should be passed by reference.
procedure AD(val DATA string, ref LIST string array, REF SIZE integer)
   SIZE <-- SIZE + 1
   LIST[SIZE] <-- DATA
endprocedure
```

==========

A bubble sort procedure. This does not contain an error, but can be made more efficient by changing : For X <-- 1 to SIZE - P

```
procedure FIX(ref LIST real array, val SIZE integer)
   declare P integer , X integer , T real
   for P <-- 1 to SIZE-1
      for X <-- 1 to SIZE-1
         if LIST[X] > LIST[X+1] then
            T <-- LIST[X]
            LIST[X] <-- LIST[X+1]
            LIST[X+1] <-- T
         endif
      endfor
   endfor
endprocedure
```

==========

A Binary Search procedure. The recursive calls are incorrect. They should not include position M.

```
procedure GET(val ITEM string, A integer, B integer)
   declare M integer
   M <-- (A + B) div 2
   if (LIST[M] = ITEM) then
       FOUND <-- M
     elsif  (ITEM > LIST[M]) then
       GET(ITEM,  M + 1  ,B)
     else
       GET(ITEM,A,  M - 1 )
   endif
endprocedure
```

==========

** Write a procedure find the LARGEST VALUE in a list of numbers.

These are all Stack and Queue algorithms.

==========

PUSH = add a new item to a stack.

```
procedure  PU(val DATA string)
   TOP <-- TOP + 1
   STACK [top] <-- DATA
endprocedure
```

==========

POP = remove an item from the stack
The parameter must be passed by reference, so the value can be returned.
Or the POP can be written as a function (the normal solution).

```
procedure PO( REF DATA string)
   DATA <-- STACK[TOP]
   TOP <-- TOP - 1
endprocedure
** State a type of error which could occur.
```

==========

EnQueue = add an item to a Queue
No subscript for DATA

```
procedure EN(val DATA string)
   TAIL <-- TAIL + 1
   LIST[TAIL] <-- DATA ~~[TAIL]~~
endprocedure
```

==========

DeQueue = Remove an item from a queue
Returns wrong value, because HEAD changes before fetching the value

```
function DE
   result string
   HEAD <-- HEAD + 1
   return LIST[HEAD-1]
endfunction
```

==========

Initialize a Stack.

```
procedure INIS
   ~~STACK~~ TOP <-- 0
endprocedure
```

==========

Initialize a Queue, must also initialize TAIL

```
procedure INIQ
   HEAD <-- 0
   TAIL <-- -1
endprocedure
```

==========

Circular EnQueue, should reset Tail to 1, not Tail-1

```
procedure EC(val DATA string)
   TAIL <-- TAIL + 1
   if TAIL > LISTMAX then
       TAIL <-- 1   ~~TAIL - 1~~
   endif
   LIST[TAIL] <-- DATA
endprocedure
```

Linked-List Algs – assume that HEAD points at the first node, but no pointer to the last node.

==========

Linear Search – it should not change TEMP if the node is found.
This way, it returns a pointer to the next node, instead of the found node.

```
function GET(val ITEM string)
   result pointer->NODE
   declare TEMP pointer->NODE, FOUND boolean

   FOUND <-- false
   TEMP <-- HEAD
   while (FOUND = false) and (TEMP # nil) do
      if (TEMP->DATA = ITEM) then
         FOUND <-- true
   ELSE
         TEMP <-- TEMP->NEXTNODE
      endif
   endwhile
   return TEMP
endfunction
```

==========

ADDS an item to the list, at the end – recursive version.
It's correct, but would be much more efficient with a TAIL pointer.

```
procedure AL(val ITEM string, ref START pointer->NODE)
   declare TEMP pointer->NODE
   if (START = nil) then
       allocate(TEMP)
       TEMP->DATA <-- ITEM
       START <-- TEMP
     else
       AL(ITEM,START->TEMP)
   endif
endprocedure          O(n)
```

==========

KILL – destroys the list and deallocates all the nodes.
Works correctly – there is no better algorithm.

```
procedure K(ref START pointer->NODE)
   declare TEMP pointer->NODE
   while (START # nil) do
      TEMP <-- START
      START <-- START->NEXTNODE
      dispose(TEMP)
   endwhile
endprocedure
```

==========

REVERSES the order of the list (backward).
The HEAD parameter must be passed by reference – otherwise the new HEAD will not be returned outside the procedure.

Tree Algorithms – assume that a binary search tree exists, with ROOT pointing to the root node
==========
This does not print all the names – the ELSE command is incorrect.
Also, the output command should be in the middle – an INORDER traversal.

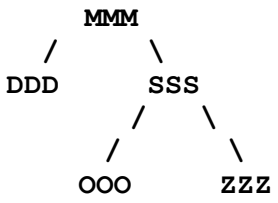
```
procedure A(val START pointer->NODE)
   if (START->LEFTCHILD # nil) then
       A(START->LEFTCHILD)
   endif

   output START->DATA

   if (START->RIGHTCHILD # nil) then
       A(START->RIGHTCHILD)
   endif
endprocedure
```

==========

B - Adds an item to the correct position in a binary search tree.
It is correct. There is no faster method, although a recursive routine is probably shorter.

==========

Counts the DEPTH of each branch of the tree. It works correctly.
The following tree prints 2, 3, 3:




```
procedure STARTC
   TREEC(ROOT,1)
endprocedure

procedure TREEC(val START pointer->NODE,val COUNT integer)
   if (START->LEFTCHILD = nil) and (START->RIGHTCHILD = nil) then
       output COUNT
     else
       if (START->LEFTCHILD # nil) then
          A(START->LEFTCHILD, COUNT+1)
       endif
       if (START->RIGHTCHILD # nil) then
          A(START->RIGHTCHILD, COUNT+1)
       endif
   endif
endprocedure
```

==========

Recursive Algorithm

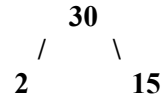
==========

This is a disaster – it contained several errors. It was supposed to print prime factors of a number. It was missing several commands. It should stop counting when X = NUM, not X > NUM.

```
main
    F(30)
endmain

procedure F(val NUM integer)
   declare X integer, FOUND boolean
   FOUND <-- false
   X <-- 2
   repeat
      if (NUM mod X = 0) then
         F(X)
         F(NUM div X)
      endif
      X <-- X + 1
   until FOUND or (X = NUM)
   if (X = NUM) then
      output NUM
   endif
endprocedure
```

==========

Trace the algorithm above – it is recursive, so the trace looks like a tree.



This won't work either – the 2 causes an infinite recursive loop. It needs a WHILE loop, not an UNTIL loop.

==========

Big O Efficiency – state the Big O efficiency of each algorithm

(a) Binary search **O(log2 N) 1000 --> 10 sec 2000 --> 11 sec**
(b) Selection sort **nested loops O(n^2)**
(c) Quick sort **O(N log N)**
(d) Sequential search **single loop O(n)**
(e) Calculating the average of a list of real numbers. **O(n)**

==========

(f) For the following algorithm, calculate the number of **iterations of the innermost loop** if the LIST array contains the following six numbers at the beginning: { 1 , 5 , 3 , 5 , 3 , 1 }

(g) State the theoretical efficiency in Big-O notation under the **worst-case scenario –** when ALL the numbers are duplicates. { 1 , 1, 1,1,1,1 }

This one is too hard, but looks like O(n^3). However, it never gets that large, because if there are a lot of duplicates they all get deleted in one pass, so it is more likely to be O(n^2). Too hard!!!

```
procedure REMOVEDUPLICATES
   declare X,D,T integer
   X <-- 1
   while (X < LISTSIZE) do
      D <-- X+1
      while (D <= LISTSIZE)
         if (LIST[D] = LIST[X]) then
             for T <-- D upto LISTSIZE
               LIST[T] <-- LIST[T+1]      How many?
             endfor
             LISTSIZE <-- LISTSIZE - 1
           else
             D <-- D+1
         endif
      endwhile
      X <-- X + 1
   endwhile
endprocedure
```

(h) Describe an O(n) algorithm for reversing the order of the elements in an array.

6.1 + 6.2 Chips

The ALU and CU are two parts of the ____ CPU ____.

The ALU performs ___ARITHMETIC__ and __ LOGIC _ operations.

The ___ CU ____ controls communications across the data bus.

The data bus contains 32 (or maybe 64) _____ PARALLEL _____ circuits,

each of which can carry one ____BIT _ of data.

The CPU uses the data bus to fetch and store data in the _____RAM _____.

But to speed up operations, the data bus does not connect directly between the CPU and the RAM.

The CPU uses the _____cache_____ for temporary storage, and **it** is connected to the RAM.

The data bus runs at a frequency of 100 __MHz____ . If the data bus width is 32 bits, then it manages

a data-transfer-rate of __ 400 __ MegaBytes per second.

The RAM can contain both __ program instructions __ and data.

The BIOS contains small utility programs, which are stored in _____ROM_____, so they are available

immediately when the computer starts.

The CPU is constantly busy executing instructions. For each instruction, it must go through the cycle of
Fetch , decode , execute , store

This could require many machine cycles. If one instruction requires 20 machine cycles, and the CPU is

running at 1 GigaHerz, the CPU could execute 1000 MHz / 20 = __**50**_____ MIPS.

Sometimes the CPU must respond to __interrupt__ signals, to take care of a request

from a __peripheral_____ device, such as the keyboard.

Every time a key is pressed on the keyboard, a signal is sent to the _ CPU _____ .

It must stop whatever it is doing and handle the request. To keep track of it's current activities, it

stores the values of all registers in a temporary storage area called the system ___ stack ____.

After it has finished "handling" the interrupt, it can ____ pop _____ all the values back out of storage and

continue where it left off.

To handle the keyboard request, the CPU accepts the ____ASCII_____ code from the keyboard,

and places this code in the keyboard buffer. This buffer is in the form of a __QUEUE__, as it must

function in a FIFO fashion. Otherwise, the order of the keystrokes would get mixed up.

6.3 + 6.4 Disk Storage

- How many sectors does a 20 GigaByte hard-disk have (if each is 512 bytes)?
* 20 Mega Kilobytes / 0.5 KB = 40 Million sectors
- How many sectors does a floppy diskette have?
* 1440 KB / 0.5 KB = 2880 sectors

- If a 16-bit number is used to store the sector ID numbers, how many sectors can there be on the disk?
* 2^16 = 65536 sectors
- If a disk has 4 billion sectors, how many bits are needed for each sector ID?
* 4 billion = 2^32, so 32 bits.

- Using a block (cluster) size of 8 KB, how much data can be stored on a disk using 16-bit codes?
* 8 KB * 2^16 = 8 * 2^6 * 2^10 KB = 512 MB
* Before Win 95, this was the storage limit for a hard-disk.

- If a disk-drive spins at 5400 RPM, and has 1024 tracks, 4 read-write heads, with a total storage capacity of 20 GB, calculate the maximum possible data-transfer-rate on best circumstances.
* 5400 RPM (revolutions per minute) = 90 Revs per Sec
* Each **CYLINDER** must contain 20 GB / 1 K = 20 MB of data. If the 4 read-write heads can transmit simultaneously, then you can get 90 * 20 MB = 1800 MB per second. There are no disk drives that can actually transmit that much. It is more likely that only 1 read/write head can be active at a time, so only ¼ of this data can be tranmsitted in each revolution, so 5 * 90 = 450 MB per sec. This is also far above the normal speed of a disk drive. Current drives can transfer 66 MB per sec under optimal conditions. This is because after 1 revolution, the track has already been read and the read/write head must **seek** to the next track, which is time consuming.

- If the disk-controller has a 32-bit data-bus running at 133 MHz, can it carry transfer the data at this theoretical maximum rate?
* 4 bytes * 133 M = 532 MB / sec. This appears to be sufficient, but as stated above this maximum theoretical speed is never reached. Indeed, the disk controller will not process data that fast anyway.

Other questions:
- Can a **virus** attack cause files to become fragmented? Justify your answer.
* No, unlikely. It might change a few bytes, but does not actually "save" the file again.
- When a file is deleted, the **data** in the **sectors** is **not changed.** What does change?
* Only the file-name entry in the directory is erased, and the sectors are marked as "available".
- Explain why it's **not possible** to change **blocking size** on a hard-disk to improve efficiency.
* The FAT (File Allocation Table) uses a fixed size for each entry – FAT32 uses 32 bits.
* If this is to be changed, the entire FAT must be restructured and rewritten, and the programs which
* read the FAT information must be reprogrammed.
- When a file is **compressed**, does the **storage structure** change, or does the **data** change?
* The DATA STRUCTURE changes. The file is also resaved, so it may well be in different sectors.
- Explain why a compressed file might load more **slowly** than a normal file.
* Decompression takes time, and so may be more time-consuming than the seek and read processes.
- Describe one other **file maintenance utility** other than defragmenter, compression, and virus scanner.
* Encryption, backup
- What would **produce** the original object modules?
* A compiler.
- If the loader is part of the OS, then what loads and executes the OS?
* The BIOS routines in the ROM load the BOOT sector, which then loads the first part of the OS.
- What does "DLL" stand for, and how does Windows manage the library modules?
* Dynamic Link Library – Dynamic because the routines are loaded when needed, and unloaded when they are no longer needed. This makes efficient use of available RAM.

6.5 Communication

Different connections use different cables. A __serial__ connections has only one single data wire. This transmits one bit of data after the next.

A ___parallel__ interface uses 8 (or more) wires to send many bits of data simultaneously.

- Which type of interface is generally faster?
* **Parallel (8 times faster)**
- State one disadvantage of the faster method.
* **Parallel wires cause electromagnetic interference**

.....
collected in a ___ BUFFER __ until it is full, or until the CPU has time to process it, store it, or send it. To speed things up, the system can use _DOUBLE BUFFERING__, where one buffer is being filled at the same time as the other as being emptied.

- Why are **interrupts** better for a keyboard rather than **polling?**
* Keyboard input is unpredictable, sometimes fast, sometimes slow, and cannot be ignored.
- Why is **polling** a suitable system for sensors (e.g. temperature sensor) rather than **interrupts?**
* Temperature data is not "urgent", and does not come at irregular intervals, and can be ignored when it is not convenient.

Controller
A separate **interface card** which sends **control signals** to the device to start or stop processes, or to monitor communication. A disk-drive controller controls the **buffering** and/or **DMA** activities.
- What does IDE mean? How can an IDE disk-drive work without needing a controller card?
* Integrated Drive Electronics – the controller card is contained in the hard disk itself.
- Explain why allowing the disk-drive to use DMA speeds up the entire system.
* No interrupts, and no waiting for buffers to fill up.
- State one **control signal** that a printer might send to the CPU.
* Out of paper, out of ink, paper jam
- State one **control signal** that might be sent from the CPU to a printer.
* Reset, start, cancel job
- State two different methods allowing a printer to differentiate between **control signals** and **data signals.**
* (1) Different ASCII codes, e.g. all codes below 32 are control codes (this is standard)
* (2) Use separate wires for control signals, rather than using the same wires as the data.

Analog/Digital
Which of the following involve DAC or ADC? Which do not?
- Storing data on a disk drive **Digital**
- Playing a phonograph record **Analog**
- Using a phone-card in a .... **Digital**
- Printing from a PC to a laser printer **Digital**
- Photography with a digital camera **ADC**
- Playing CD music on a PC speaker **DAC**
- Measuring temperatures and storing them **ADC**
- Controlling an industrial robot **DAC**
- Audio sampling (what's that?) **ADC**
- Scanning and OCR **ADC**

Which is more "realistic" – analog or digital data? **Analog**
Which can be copied more perfectly – analog or digital data? **Digital (perfect copies, and easy)**
Which requires more storage space – analog or digital data? **Analog**
Which can be transmitted more rapidly across the Internet – analog or digital data? **Digital**

7 System Life Cycle

Why is it a life **cycle?** What about it is cyclic?
* Computer systems are designed and created, but later they may be changed, redesigned, and rebuilt

Stage	Description
Analysis	Deciding what data needs to be stored and features need to be implemented **Design**
Design	Installing and debugging the finished system **Operation**
Implementation	Investigating the existing system, talking to the intended users **Analysis**
Operation	Writing programs and/or building hardware **Implementation**
Maintenance	Making minor changes to adjust to new requirements, fixing problems **Maintenance**

* Sample data guides the design. A lot of text requires a bigger box on the screen. Customer info needs specific fields (name, address, phone) and the programmer needs to know what is required. Not collecting sample data usually results in programs with missing or unusable features.
* **requirements specification –** a list of required FEATURES and FUNCTIONS.
***feasibility study –** will it be possible, cheap enough, and practical
* The feasibility is usually done first (in Analysis) before the requirements are written (design)
* **module level –** testing individual procedures and functions
* **system testing –** testing the entire finished system
* **Test data =** the data which will be entered
* **Test case =** test data + expected result
* **Failed systems, incorrect results, lost money, lost time**
* Positive – save money, cheaper (and perhaps better) products
  Negative – lost jobs, sometimes worse or unreliable products

Long Question - Video shop
*(a) Customer data, film data, price data
*(b) They may have preferences of how and what they want to use, they may be worried about price, they may have a supplier who can only supply specific type of equipment.
*(c) No expertise in programming
*(d) He/she doesn't really know much about the user's needs
*(e) Might need fewer employees, might need employees with different skills
*(f) Bar-code reader might be more efficient than keyboard input of code numbers
*(g) Printer for printing receipts and paper reports
*(h) On-line – when a video is returned, it should immediately be "checked-in" so it can be checked out again. Also, clerks might search for a film in a database if they don't know the title.
*(i) It is unlikely that a catastrophe occurs, but equipment failure might lose the records of the videos currently checked out, so it is possible that some videos might be lost.
*(j) Daily, after closing