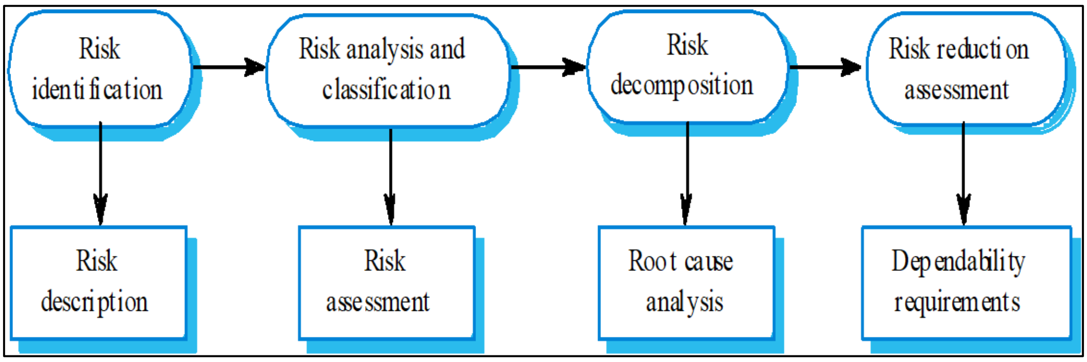
Software Engineering II

Lecture 2

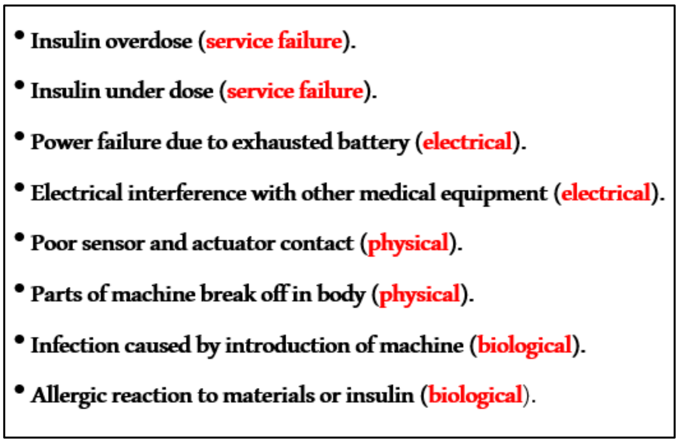
- **Stages of risk-based analysis:**
  1. Risk identification: Identify possible risks.
  2. Risk analysis and classification: Calculate the seriousness of each risk.
  3. Risk decomposition: Decompose risks to discover their source.
  4. Risk reduction assessment: Define how each risk must be taken into removed or reduced when the system is designed.



- **Dependability Requirements:**
  1. Functional requirements to define:
     - error checking
     - recovery facilities
     - protection against system failures.
  2. Non-functional requirements defining the required reliability and availability of the system.
  3. Excluding requirements that define states and conditions that must not arise.

1. Risk Identification:

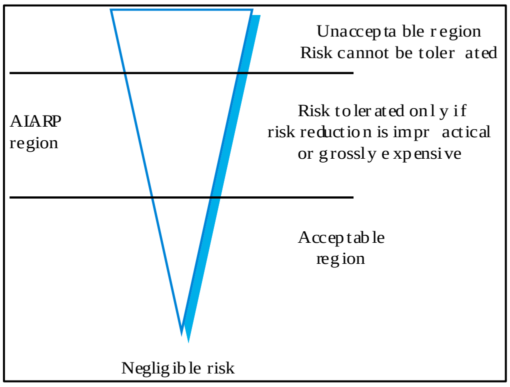
- Identify the risks faced by the critical system.
- In safety-critical systems, the risks are the risks that lead to accidents.
- In security-critical systems, the risks are the possible attacks.
- In risk identification, you should identify risk classes and position risks in these classes (Service failure, Electrical risks, etc.)
- Ex: **Insulin pump risks**



- Insulin overdose (service failure).
- Insulin under dose (service failure).
- Power failure due to exhausted battery (electrical).
- Electrical interference with other medical equipment (electrical).
- Poor sensor and actuator contact (physical).
- Parts of machine break off in body (physical).
- Infection caused by introduction of machine (biological).
- Allergic reaction to materials or insulin (biological).

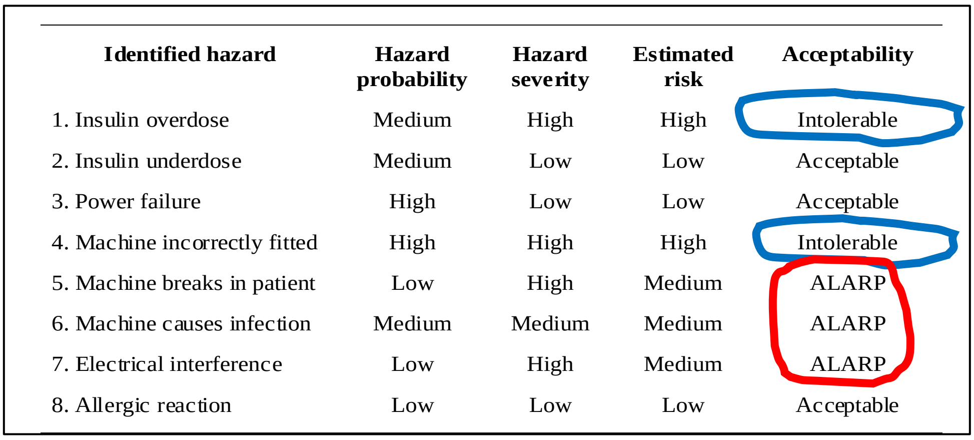
2. Risk analysis and classification:

- understanding the probability that a risk will arise and the possible values if an accident or incident should occur.
- Risks may be categorized as:
  1. Intolerable: must never arise (we could not live with!)
  2. As Low As Reasonably Practical (ALARP): must minimize the possibility of risk given cost and schedule constraints (we have to face it!)
  3. Acceptable: the risk is acceptable and no extra costs (we can to live with!)



- The acceptability of a risk is determined by human, social and political considerations
- Risk assessment is subjective (personal)
- Risk assessment: The risk probability, The risk severity
- The aim is to reject risks that are likely to arise or have high severity

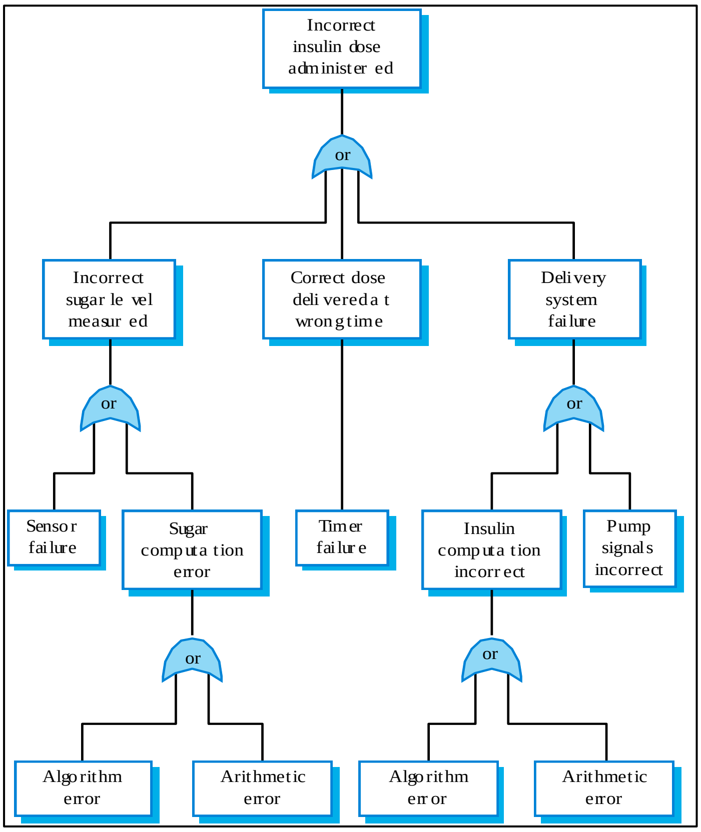
- Ex: **Risk assessment - insulin pump**



Identified hazard	Hazard probability	Hazard severity	Estimated risk	Acceptability
1. Insulin overdose	Medium	High	High	Intolerable
2. Insulin underdose	Medium	Low	Low	Acceptable
3. Power failure	High	Low	Low	Acceptable
4. Machine incorrectly fitted	High	High	High	Intolerable
5. Machine breaks in patient	Low	High	Medium	ALARP
6. Machine causes infection	Medium	Medium	Medium	ALARP
7. Electrical interference	Low	High	Medium	ALARP
8. Allergic reaction	Low	Low	Low	Acceptable

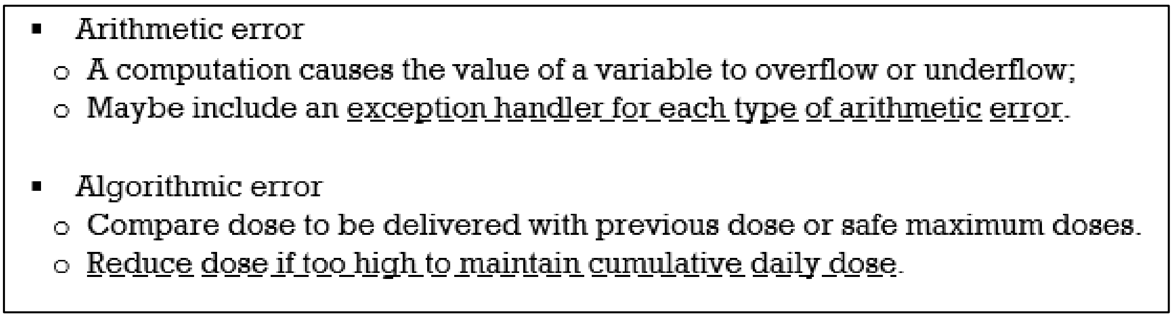
3. Risk decomposition:

- Concerned with discovering the root causes of risks.
- 2 Techniques:
  1. Inductive, bottom-up techniques. Start with a proposed system failure and measure the hazards that could arise from that failure.
  2. Deductive, top-down techniques. Start with a hazard and deduce what the causes of this could be.
- Fault-tree analysis (deductive top-down technique):
  - Put the risk at the root of the tree and identify the system states that lead to that hazard.
  - Where appropriate, link these with **'and'**, **'or'** conditions.
  - Goal to minimize the number of single causes of system failure.
- Ex: **Insulin pump fault tree**



4. Risk reduction assessment:

- The aim is to identify dependability requirements that specify how the risks should be managed and ensure that accidents do not arise.
- Risk reduction strategies
  - Risk avoidance
  - Risk detection and removal
  - Damage limitation
- a mix of risk reduction strategies are used.
- Ex: **Insulin pump-Risk reduction assessment**



- Arithmetic error
  - A computation causes the value of a variable to overflow or underflow;
  - Maybe include an exception handler for each type of arithmetic error.

- Algorithmic error
  - Compare dose to be delivered with previous dose or safe maximum doses.
  - Reduce dose if too high to maintain cumulative daily dose.

- Example: **chemical plant control system**
  - The system will include sensors to detect and correct excess pressure in the reactor.
  - However, it will also include an independent protection system that opens a break controller if dangerously high pressure is detected.

- **Safety specification (against accidents):**
  - Safety requirements usually apply to the system rather than to individual sub-systems.
  - Ex: **Safety requirements - insulin pump**

SR1: The system shall not deliver a single dose of insulin that is greater than a specified maximum dose for a system user.

SR2: The system shall not deliver a daily cumulative dose of insulin that is greater than a specified maximum for a system user.

SR3: The system shall include a hardware diagnostic facility that shall be executed at least 4 times per hour.

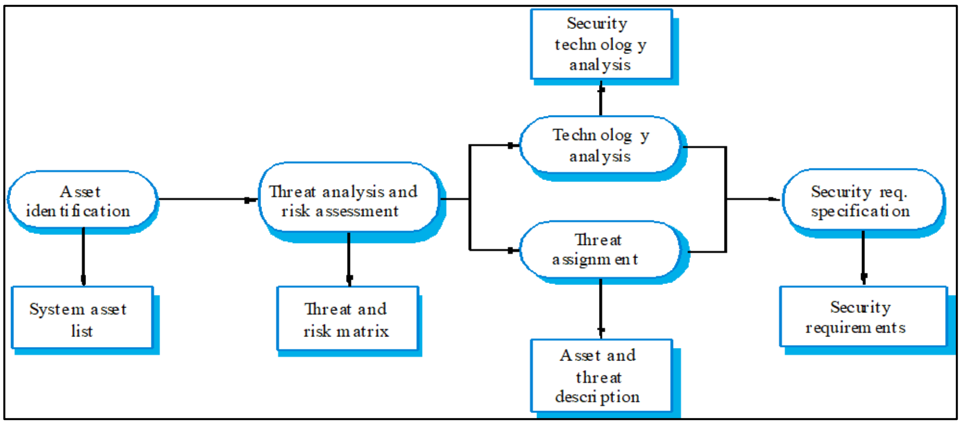
SR4: The system shall include an exception handler for all of the exceptions that are identified in Table 3.

SR5: The audible alarm shall be sounded when any hardware or software anomaly is discovered and a diagnostic message as defined in Table 4 should be displayed.

SR6: In the event of an alarm in the system, insulin delivery shall be suspended until the user has reset the system and cleared the alarm.

  - Safety requirements types:
    - Functional safety requirements
      - The define how the system should provide protection.
    - Safety integrity requirements
      - These define the reliability and availability of the protection system. They are based on expected usage and are classified using a safety integrity level from 1 to 4.

- **Security specification (against attacks):**
  - Differences between safety:
    - No well-defined notion of a security life cycle.
    - Generic threats rather than system specific hazards.
    - Complete security technology, but there are problems in transferring this into general use.
    - The dominance of a single supplier means that huge numbers of systems may be affected by security failure.
  - Stages in security specification:
    1. Asset identification and evaluation
       - The assets and their required degree of protection are identified.
    2. Threat analysis and risk assessment
       - Possible security threats are identified, and the risks associated.
    3. Threat assignment
       - For each identified asset, there is a list of associated threats.
    4. Technology analysis
       - Available security technologies are assessed.
    5. Security requirements specification
       - The security requirements are specified.


Security technology analysis
Technology analysis
Asset identification
Threat analysis and risk assessment
Threat assignment
Security req. specification
System asset list
Threat and risk matrix
Asset and threat description
Security requirements

- Types of security requirement:
  - Identification requirements.
  - Authentication (validation) requirements.
  - Authorization requirements.
  - Immunity requirements.
  - Integrity (completeness) requirements.
  - Intrusion detection requirements.
  - Privacy requirements.
  - Security auditing requirements.
  - System maintenance security requirements.

- Ex: **LIB-SYS security requirements**

SEC1: All system users shall be identified using their library card number and personal password.

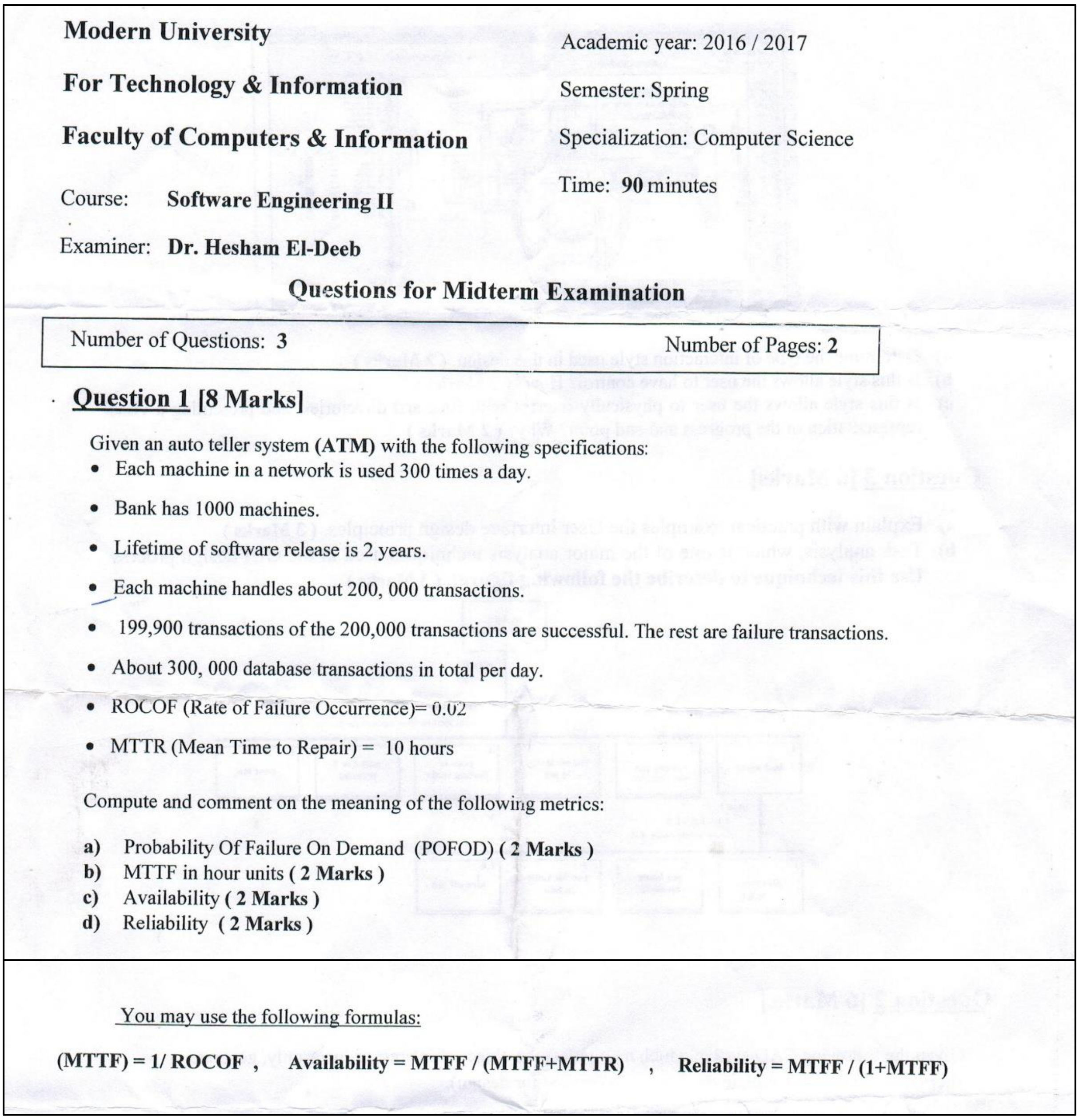
SEC2: Users privileges shall be assigned according to the class of user (student, staff, library staff).

SEC3: Before execution of any command, LIBSYS shall check that the user has sufficient privileges to access and execute that command.

SEC4: When a user orders a document, the order request shall be logged. The log data maintained shall include the time of order, the userÕs identification and the articles ordered.

SEC5: All system data shall be backed up once per day and backups stored off-site in a secure storage area.

SEC6: Users shall not be permitted to have more than 1 simultaneous login to LIBSYS.



Modern University

For Technology & Information

Faculty of Computers & Information

Course: **Software Engineering II**

Examiner: **Dr. Hesham El-Deeb**

Academic year: 2016 / 2017

Semester: Spring

Specialization: Computer Science

Time: **90** minutes

Questions for Midterm Examination

Number of Questions: **3** Number of Pages: **2**

Question 1 [8 Marks]

Given an auto teller system **(ATM)** with the following specifications:

- Each machine in a network is used 300 times a day.
- Bank has 1000 machines.
- Lifetime of software release is 2 years.
- Each machine handles about 200, 000 transactions.
- 199,900 transactions of the 200,000 transactions are successful. The rest are failure transactions.
- About 300, 000 database transactions in total per day.
- ROCOF (Rate of Failure Occurrence)= 0.02
- MTTR (Mean Time to Repair) = 10 hours

Compute and comment on the meaning of the following metrics:

a) Probability Of Failure On Demand (POFOD) **(2 Marks)**
b) MTTF in hour units **(2 Marks)**
c) Availability **(2 Marks)**
d) Reliability **(2 Marks)**

You may use the following formulas:

(MTTF) = 1/ ROCOF , Availability = MTFF / (MTFF+MTTR) , Reliability = MTFF / (1+MTFF)

Solution

- **Rate of Fault Occurrence (ROCOF):**
  - the rate of occurrence of failure in the system.
  - ROCOF of 0.02 means 2 failures are likely in each 100-operational hour.
  - Appropriate for operating systems, transaction processing systems
  - The value is needed to be small as we can to achieve reliability.

a. **Probability of failure on demand (POFOD):**

- the probability that the system will fail when a service request is made.
- POFOD = $\frac{200000-199900}{200000} \times 100$ = 0.05
- That mean that 5 of 100 service request may result in fail.
- Appropriate for protection systems
- The value is needed to be small as we can to achieve reliability.

b. **MTTF (Mean time to failure):**

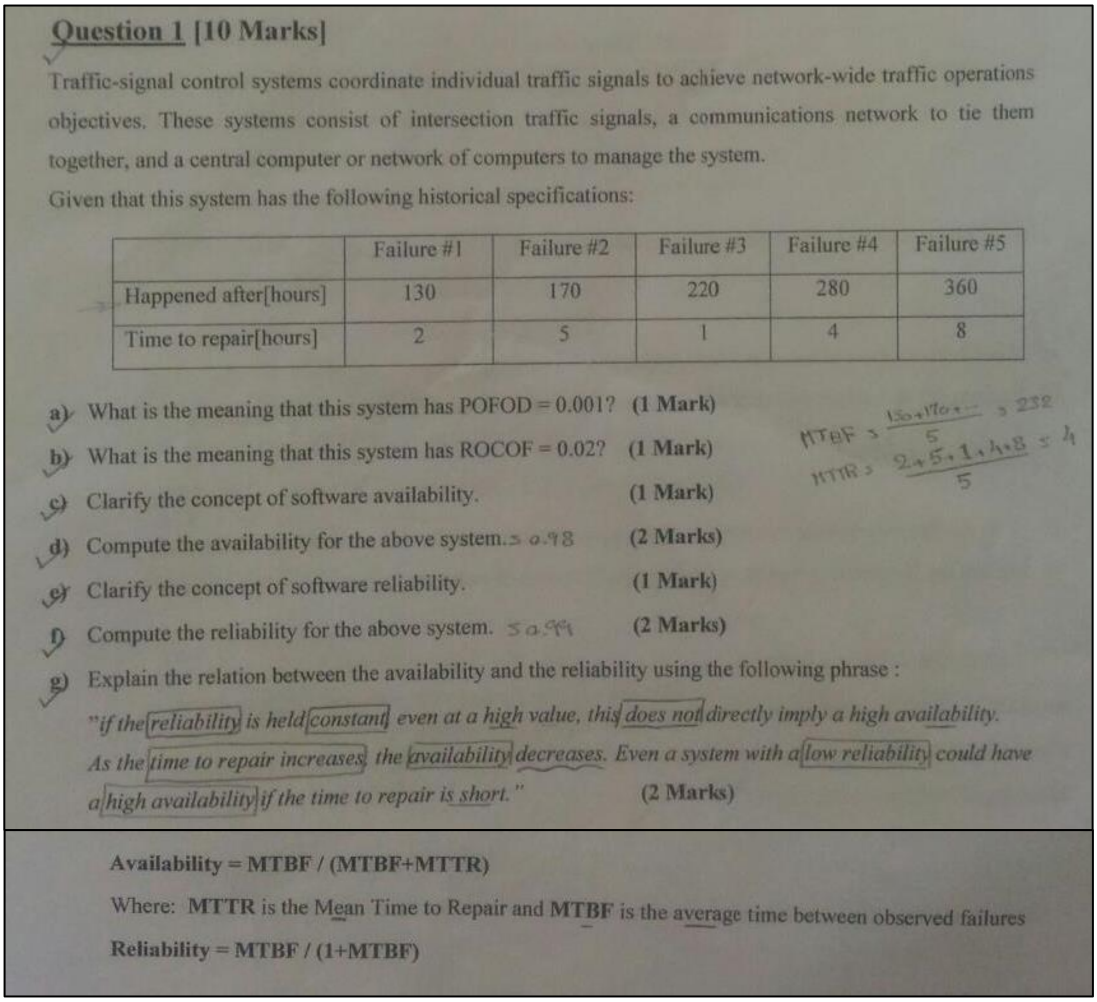
- the average time between observed failures of the system.
- MTTF = $\frac{1}{\text{ROCOF}} = \frac{1}{0.02} =$ 50 hour
- Appropriate for systems with long transactions.
- The value is needed to be high as we can to achieve reliability.

c. **Availability:**

- Measure of the fraction of the time that the system is available for use, takes repair and restart time into account
- Availability = $\frac{\text{MTTF}}{\text{MTTF+MTTR}} = \frac{50}{50+10}$ = 0.833
- Availability of 0.833 mean software is available for 833 out of 1000 hour.
- Appropriate for non-stop, continuously running systems.
- The value is needed to be high as we can to achieve reliability.

d. **Reliability:**

- The probability that an item will perform a required function, under stated conditions, for a stated period
- Reliability = $\frac{\text{MTTF}}{\text{1+MTTF}} = \frac{50}{1+50}$ = 0.980
- Reliability of 0.980 mean software is reliable for 980 out of 1000 hour.
- The value is needed to be high as we can, the ideal case is 1, so as we get towards 1 is best.



Question 1 [10 Marks]

Traffic-signal control systems coordinate individual traffic signals to achieve network-wide traffic operations objectives. These systems consist of intersection traffic signals, a communications network to tie them together, and a central computer or network of computers to manage the system.

Given that this system has the following historical specifications:

	Failure #1	Failure #2	Failure #3	Failure #4	Failure #5
Happened after[hours]	130	170	220	280	360
Time to repair[hours]	2	5	1	4	8

a) What is the meaning that this system has POFOD = 0.001? **(1 Mark)**

b) What is the meaning that this system has ROCOF = 0.02? **(1 Mark)**

c) Clarify the concept of software availability. **(1 Mark)**

d) Compute the availability for the above system. = 0.98 **(2 Marks)**

e) Clarify the concept of software reliability. **(1 Mark)**

f) Compute the reliability for the above system. = 0.99 **(2 Marks)**

g) Explain the relation between the availability and the reliability using the following phrase :

"if the reliability is held constant even at a high value, this does not directly imply a high availability. As the time to repair increases, the availability decreases. Even a system with a low reliability could have a high availability if the time to repair is short." **(2 Marks)**

MTBF = $\frac{130+170+\cdots}{5}$ = 232

MTTR = $\frac{2+5+1+4+8}{5}$ = 4

Availability = MTBF / (MTBF+MTTR)

Where: **MTTR** is the Mean Time to Repair and **MTBF** is the average time between observed failures

Reliability = MTBF / (1+MTBF)

a. POFOD oof 0.001 means that 1 of 1000 service request may result in fail.

b. ROCOF of 0.02 means 2 failures are likely in each 100-operational hour.

c. Measure of the fraction of the time that the system is available for use, takes repair and restart time into account

d. MTBF = $\frac{(130-0)+(170-130)+(220-170)+(280-220)+360-280)}{5}$ = 72

MTTR = $\frac{2+5+1+4+8}{5}$ = 4

Avalibality = $\frac{72}{72+4}$ = 0.95

e. The probability that an item will perform a required function, under stated conditions, for a stated period

f. Reliablilty = $\frac{72}{1+72}$ = 0.99

g. The time to repair is the key factor of determine the avalibality of the sysetm even the reliablilty is low

- **Failure Avoidance:**

A. Programmer Measures: (Q: what programmer do to achieve reliability)

1) Recursion (not recommended):
   - Ex: Errors in recursion can cause memory overflow.
   - Practical Ex:

```
int factorial (int n) {
        if (n <= 1)
              return 1;
        else
              return n * factorial (n-1);
      }
```

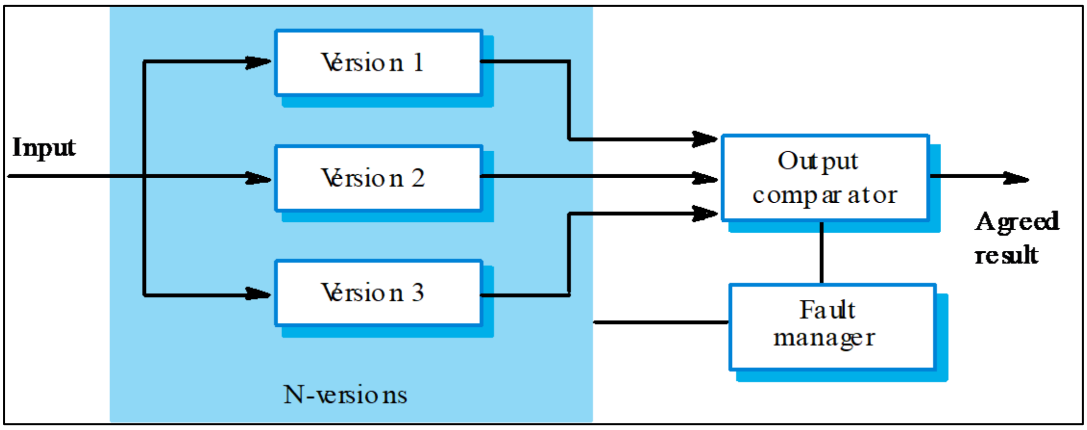
2) Pointers (not recommended):
   - Pointers referring to the wrong memory areas can corrupt data.
   - Ex: Using more than 1 name to refer to the same state variable can make programs difficult to understand and change.
   - Practical Ex:

```
int number = 88;
int * pNumber;
pNumber = &number;
int * pAnother = &number;
```

3) Dynamic memory allocation (not recommended):
   - Ex: Run-time allocation can cause memory overflow.
   - Practical Ex:

```
int numbers [5];
```

4) N-version programming (highly recommended):
- The same specification is implemented in many different versions by different teams.
- All versions compute at the same time, and the majority output is selected using a voting system.



- The output comparator is a simple piece of software that uses a voting mechanism to select the output.

5) Consistency Checks (recommended-Runtime Detection):
- identify a bad computational result.
- exploit characteristics of data to identify problems.
- parallel dissimilar computation for result comparison.
- recovery strategy required
- Ex: protect against addressing and synchronization failures

6) Watchdog Timers (WDT) (recommended-Runtime Detection):
- Device card that performs a specific operation after a certain period if something goes wrong with an electronic system and the system does not recover on its own.
- A watchdog timer can be programmed to perform a warm boot (restarting the system) after a certain number of seconds during which a computer fails to respond the most recent action.
- contains a digital counter that counts down to zero. If the counter reaches zero before the computer recovers, a signal is sent to designated circuits to perform the desired action.
- require hardware to support interrupt tasks.
- WDT timer causes status check routine to be called.
- status check routine verify that code is doing what required.
- recovery strategy required.
- Ex: protect against runway control flow failure.

7) Bounds Checking (recommended-Runtime Detection):
   - compare results of computation with known bounds to identify bad results.
   - can't protect against bad result which has reasonable value.
   - recovery strategy required.
   - Ex: protect against numerical failure or propagated numerical failure

----

B. Designer Measures:

1. For each sub-system, analyse the consequences of possible system failures.
2. From the system failure analysis, partition failures into appropriate classes.
3. For each failure class identified, set out the reliability using an appropriate metric. Different metrics may be used for different reliability requirements.
4. Identify functional reliability requirements to reduce the chances of critical failures.

Lecture 3

- **Human factors in interface design**
  1. Limited short-term memory
     People can remember about 7 items of information.
     If you present more than this, they are more liable to make mistakes.
  2. People make mistakes
     When people make mistakes and systems go wrong, inappropriate alarms and messages can increase stress and hence the probability of more mistakes.
  3. People are different
     People have a wide range of physical capabilities.
     Designers should not just design for their own capabilities.
  4. People have different interaction preferences
     Some like pictures, some like text.

UI Design Issues

- UI design must take account of the needs, experience and capabilities of the system users.
- Designers should be aware of people's physical and mental limitations and should recognise that people make mistakes.
- UI design principles based on interface designs although not all principles are valid to all designs.

1. User interface design principles

1. User familiarity	The interface should have terms and concepts familiar with the users rather than computer. **For example**, an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.
2. Consistency	The system should display an appropriate level of consistency. **For example**, Commands and menus should have the same format, command punctuation (stop, numbering) should be similar, etc.
3. Minimal surprise	Users should never be surprised by the behaviour of the system. If a command operates in a known way, the user will be able to predict the operation of similar commands.
4. Recoverability	The system should provide some flexibility to user errors and allow the user to recover from errors. This might include an undo, confirmation of damaging actions.

5. User guidance	Some user guidance such as help systems should be supplied
6. User diversity	Interaction facilities for different types of user should be supported. **For example**, some users have seeing difficulties and so larger text should be available

2. Design Problems in UIs

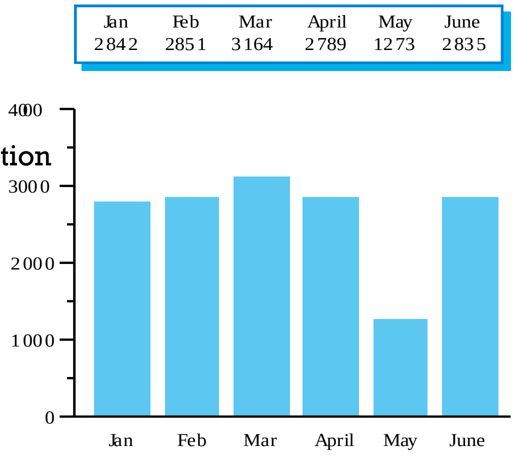
1. How should information from the user be **provided** to the computer system?
2. How should information from the computer system be **presented** to the user?
   - User interaction and information presentation may be integrated

3. Interaction Styles

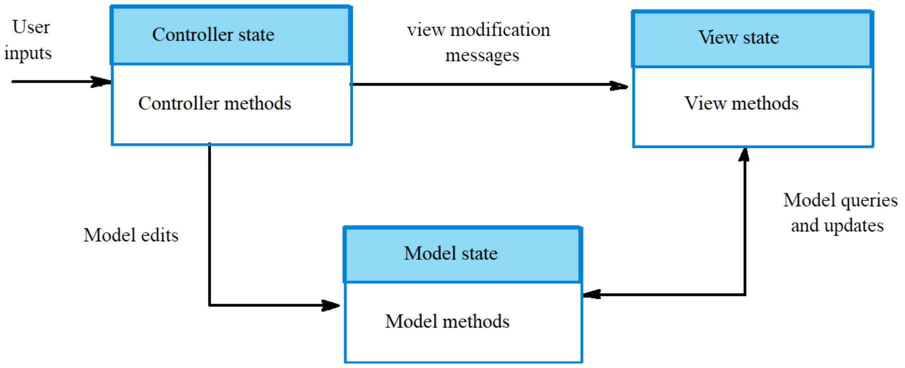
Style	Adv	Disadv	App
1. Direct manipulation	Fast interaction Easy to learn	Hard to implement Suitable for visual symbol for tasks and objects	Video games CAD systems
2. Menu selection	Avoid user errors Little typing required	Slow for experienced users Can become complex if many menus options	Most general-purpose systems
3. Form fill-in	Simple data entry Easy to learn Checkable	Take a lot of screen space Cause problems where user don't match the form fields	Stock control Personal loan system
4. Command language	Powerful and flexible	Hard to learn Poor error management	Operating system Command and control system
5. Natural language	Accessible to casual user Easily extended	Require more typing Its understanding system aren't reliable	Information retrieval system

4. Information Presentation

- **The information may be presented:**
  1. Directly (e.g. text in a word processor)
  2. Transformed in a way for graphical presentation



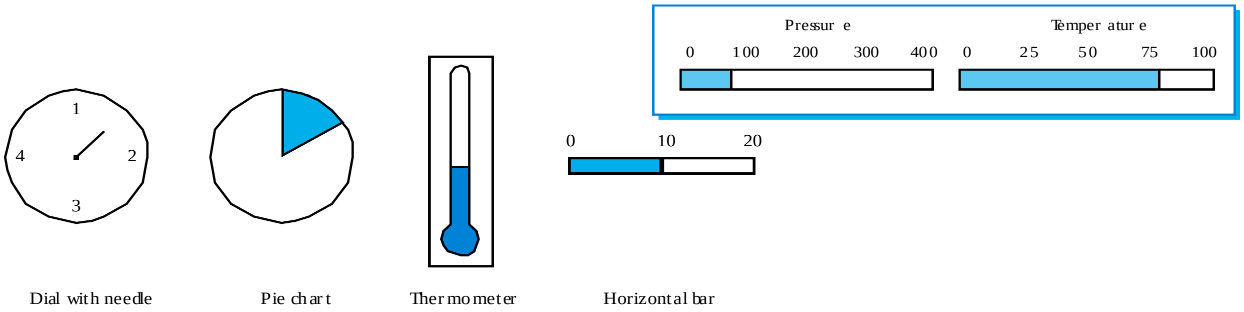
- **The Model-View-Controller approach** is a way of supporting multiple presentations of data.



- **Information Presentation:**
  1. Static information
     - Initialised at the beginning of a session.
     - It does not change during the session.
     - May be either numeric or textual.
  2. Dynamic information
     - Changes during a session
     - Changes must be communicated to the system user.
     - May be either numeric or textual.

- **Information Display Factors:**
  - Is the user interested in precise information or data relationships?
  - How quickly do information values change? Must the change be indicated immediately?
  - Must the user take some action in response to a change?
  - Is there a direct manipulation interface?
  - Is the information textual or numeric? Are relative values important?

- **Analogue or digital presentation?**
  1. Digital presentation
     - Compact - takes up little screen space
     - Precise values can be communicated.
  2. Analogue presentation
     - Easier to get an 'at a glance' impression of a value
     - Possible to show relative values
     - Easier to see exceptional data values (odd values).



5. Data Visualization

- Concerned with techniques for displaying large amounts of information.
- Visualisation can express relationships between entities and directions.
- **Examples:**
  1. Weather information.
  2. Telephone network.
  3. Chemical plant visualised by showing pressures and temperatures.
  4. A model of a molecule (group of atoms) displayed in 3 dimensions.
  5. Web pages displayed as a hyperbolic tree.

6. Colour Displays

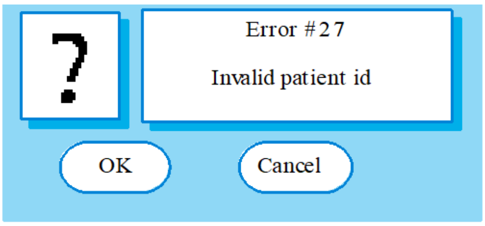
- Colour help the user understand complex information structures.
- Colour can be used to highlight exceptional event.
- **Common mistakes:**
  1. The use of colour to communicate meaning.
  2. The over-use of colour in the display.

- **Colour Use Main Guidelines:**
  1. **Limit the number** of colours used and be conservative in their use.
  2. Use **colour change** to show a change in system status.
  3. Use **colour coding** to support the task that users are trying to perform.
  4. Use colour coding in a **consistent** way.
  5. Be careful about **colour pairings**.

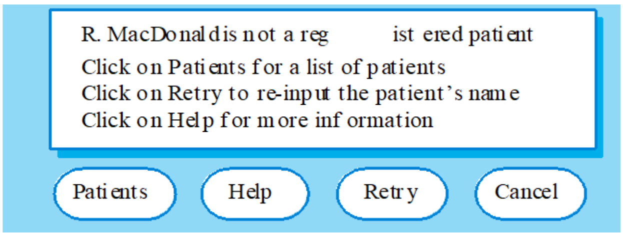
- **Murch's Rules:**
  1. Avoid pure BLUE.
  2. Avoid family of BLUE.
  3. Older users need higher brightness levels to distinguish colours.
  4. It is difficult to focus by colour alone, it's just a tool to help.
  5. Avoid RED and GREEN on large-scale displays.
  6. Opposite colours go well together (Match colours)
  7. Avoid single-color, for colour blind observers
  8. Avoid highly extreme colours.

7. Error Messages

- Error message design is critically important.
- Messages should be polite, concise, consistent and constructive.
- The background and experience of users should be determined.


Error #27

Invalid patient id

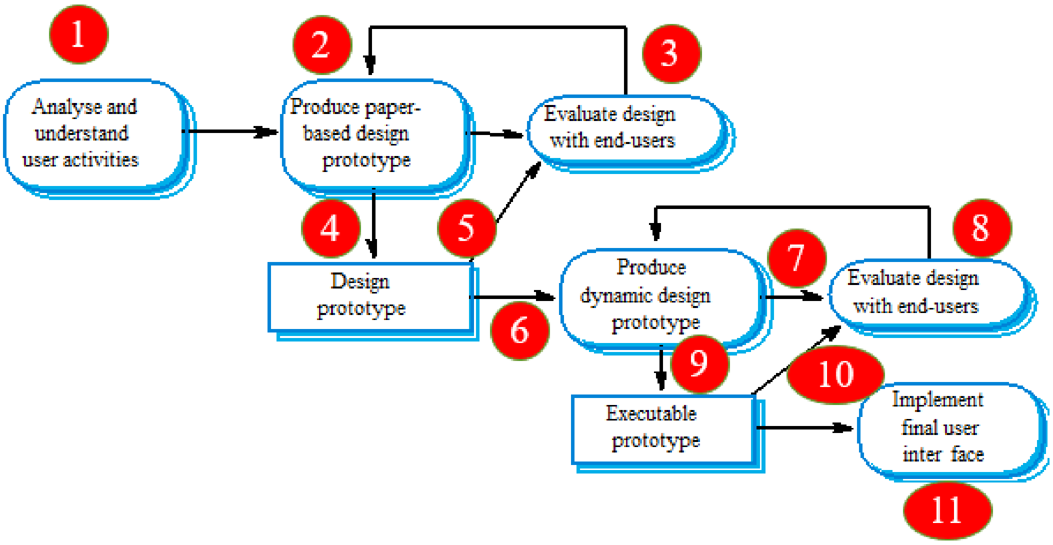

R. MacDonald is not a registered patient
Click on Patients for a list of patients
Click on Retry to re-input the patient's name
Click on Help for more information

- **Design factors in message wording:**

1. Context	The message should reflect the current user context The system should aware of what the user is doing and generate message related to the current activity.
2. Experience	Provide 2 types for message for beginners and old users and allow them to control message they want.
3. Skill level	The message should be suitable for the user skills and experiences
4. Style	Message should be positive rather than negative. Use the active rather than passive mode of address. Never be rude or try to be funny.
5. Culture	The message should be familiar with the culture of the country where the system is sold. A suitable message for one culture might be unacceptable in another.

The UI Design Process

- UI design is an iterative process involving close relations between users and designers.
- The 3 core activities in this process are:
  1. **User analysis:**
     Understand what the users will do with the system
  2. **System prototyping:**
     Develop a series of prototypes for experiment
  3. **Interface evaluation:**
     Experiment with these prototypes with users.


1
Analyse and understand user activities
2
Produce paper-based design prototype
3
Evaluate design with end-users
4
5
Design prototype
6
Produce dynamic design prototype
7
8
Evaluate design with end-users
9
10
Executable prototype
Implement final user inter face
11

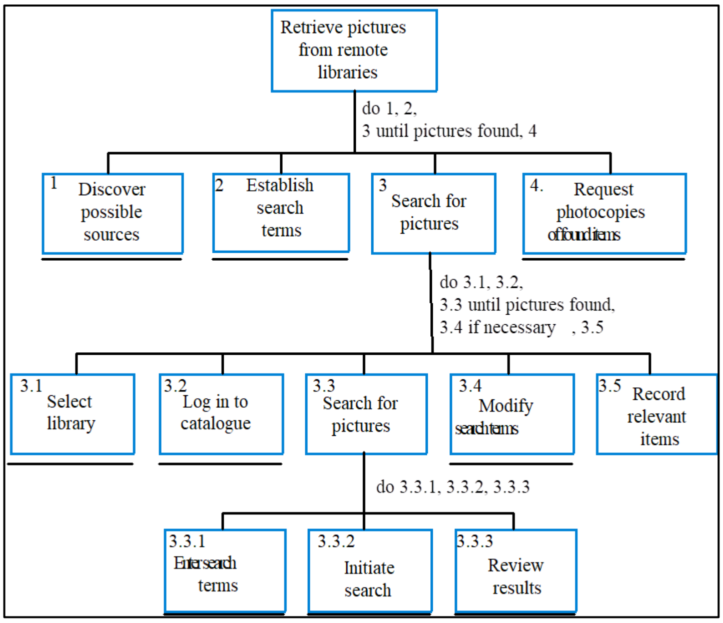
1. User Analysis

- If you don't understand what the users want to do with a system, you have no realistic prospect of designing an effective interface.
- User analyses must be described in terms that users and other designers can understand.
- **Scenarios** where you describe typical probabilities of use, are one way of describing these analyses.

- **Analysis Techniques:**

1. Task analysis

- Model the steps involved in completing task by a hierarchical model.



Retrieve pictures from remote libraries
do 1, 2, 3 until pictures found, 4
1 Discover possible sources
2 Establish search terms
3 Search for pictures
4. Request photocopies of found items
do 3.1, 3.2, 3.3 until pictures found, 3.4 if necessary , 3.5
3.1 Select library
3.2 Log in to catalogue
3.3 Search for pictures
3.4 Modify search terms
3.5 Record relevant items
do 3.3.1, 3.3.2, 3.3.3
3.3.1 Enter search terms
3.3.2 Initiate search
3.3.3 Review results

2. Interviewing and questionnaires

- Asks the users about the work they do.
- **Interviewing Issues:**
  - Design semi-structured interviews based on open-ended questions.
  - Users can then provide information that they think is essential (highlight spot), not just information that you have thought of collecting.
  - Group interviews or focus groups allow users to discuss with each other what they do.

3. Ethnography (online observation)

- Observes the user at work.
- Involves an external observer watching users and questioning them in an unscripted way about their work.
- Valuable because many user tasks are intuitive, and they find these **very difficult to describe and explain**.
- Also helps understand the role of social and organizational influences on work.

2. User interface prototyping

- The aim is to allow users to gain direct experience with the interface.
- Without it, it is impossible to judge the usability of an interface.

- **Prototyping may be a two-stage process:**
  1. **Paper.**
     - Work through scenarios using sketches of the interface.
     - Use a storyboard to present a series of interactions with the system.
     - It is an effective way of getting user reactions to a design proposal.

  2. **Automated.**
     - **Script-driven prototyping**
       Develop a set of scripts and screens using a tool.
       When the user interacts with, the screen change to the next display.
     - **Visual programming**
       Use a language designed for rapid development (Visual Basic).
     - **Internet-based prototyping**
       Use a web browser and associated scripts.

3. User Interface Evaluation

- Evaluation of interface design should be carried out to assess its suitability.
- Full scale evaluation is very expensive and impractical for most systems.
- Ideally, an interface should be evaluated against a **usability specification**

1. Learnability	How long does it take a new user to become productive with the system?
2. Speed of operation	How well does the system response match the user practice?
3. Robustness	How tolerant is the system of user error?
4. Recoverability	How good is the system at recovering from user errors?
5. Adaptability	How closely is the system tied to a single model of work?

However, it is rare for such specifications to be produced.

- **Simple Evaluation Techniques:**
  1. Questionnaires for user feedback.
  2. Video recording of system use.
  3. Arrangement of code to collect information about facility use and errors.
  4. The ability to collect on-line user feedback

User Interface Bad Design Tips

1. Forgetting the User

- Developers, often design for what they know, not what the user knows
- Problem occurs in other areas of development (testing, documentation)
- It is critical in the interface because it makes the user feel incapable of using the product.

2. Failing to Give the User Control

- The designers prefer to control user navigation by graying and blackening menu items or controls within an application.
- Controlling the user is completely inconsistent to event-driven design in which the user, rather than the software, dictates what events occur.
- As a developer, if you are spending a lot of time dynamically graying and blackening controls, you need to re-examine your design approach and realize that you may be controlling the user, who may not want to be controlled.

3. Providing Too Many Features at the Top Level

- VCR 1985 and 1995:
- In 1985 have a plenty of buttons readily available on the faceplate.
- In 1995 have only a few buttons for the key features that people use: play, fast forward, reverse, stop, and eject.
- 1995 have even more features than 1985, yet the features will be behind a drop-down panel or sliding door, accessible when needed but not staring the user in the face.

User Interface Good Design Tips

1. Understand People

- Applications must reflect the perspectives and behaviors of their users.
- People learn more easily by recognition than by recall, the average person can recall about 2,000 to 3,000 words, yet can recognize more than 50,000 words.
- Always attempt to provide a list of data values from which the user can select, rather than having the user key in values from memory.

2. Be Careful of Different Perspectives

- Many designers fall into the perspective trap when it comes to icon design or the overall behavior of the application.

3. Design for Clarity GUI

- Applications often are not clear to end users, common complaint among users is that certain terms are not clear.
- **Ex:** When the application is released, 3 different screens has the name of “Item, Product, Merchandise", when all three terms denote the same thing.
- This lack of consistency leads to confusion for users
- One effective way to increase the clarity of an application is to develop and use a list of reserved words.

4. Design for Consistency with other most popular applications

- Good GUIs use consistent behavior throughout the application and build upon a user's previous knowledge of other successful applications
- **Ex:** The bottom line is that business travelers are not looking for a new and exciting experience in each new city. Business users of your software have similar needs
- Each new and exciting experience you provide in the software can become an anxiety-inducing experience or expensive.

5. Be Careful with Audible

- Audible feedback can be useful in cases where you need to warn the user of a serious problem, but sometimes it’s annoying sounds
- Allow users to disable audio feedback, except in cases when an error must be addressed.

6. Keep Text Clear

- Developers often try to make textual feedback clear by adding a lot of words, but they make the message less clear.
- Textual feedback can be handled most effectively by assigning these tasks to experienced technical writers (professional editors).

7. Provide Traceable Paths

- Providing a traceable path is harder than it sounds.
- You must also identify areas where you can flatten the menu structure and avoid more than two levels of cascading menus.
- Providing a descriptive title bar within each dialog box helps to remind users what menu items or buttons were pressed to bring them in focus.

8. Provide Keyboard Support

- Keyboards are a common feature on users' desktops and provide an efficient means to enter text and data.
- Using a mouse can become time-consuming and inefficient for the touch typist or frequent users of an application.
- The keyboard accelerators should be easy to access and limited to one or two keys (such as F3 or Ctrl-P).

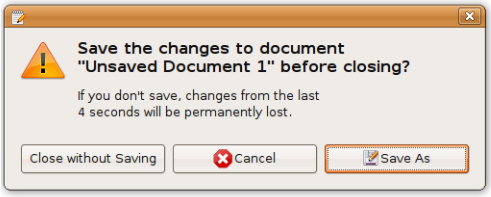
- Keyboards have limitations in the GUI, such as when trying to implement direct-manipulation tasks like drag and drop, pointing, and re-sizing.
- So, provide complete and equal keyboard and mouse support for all menu and window operations.

9. Watch the Presentation Model

- A critical aspect that ties all these facts of the interface together is the interface's look and feel.
- The model should involve careful decisions, such as whether the application has a single or multiple document interface.
- On the other hand, if you do not define the presentation model early in the design, late changes to the look and feel of the application will be much more costly and time-consuming because nearly every window may be affected.

10. Use Modal vs. Modeless Dialogs Appropriately

- When we output or input from the user, we often use a **modal** dialog box.
- **Modal** dialogs have many uses in complex applications since most people only work on one window at a time.



Save the changes to document "Unsaved Document 1" before closing?

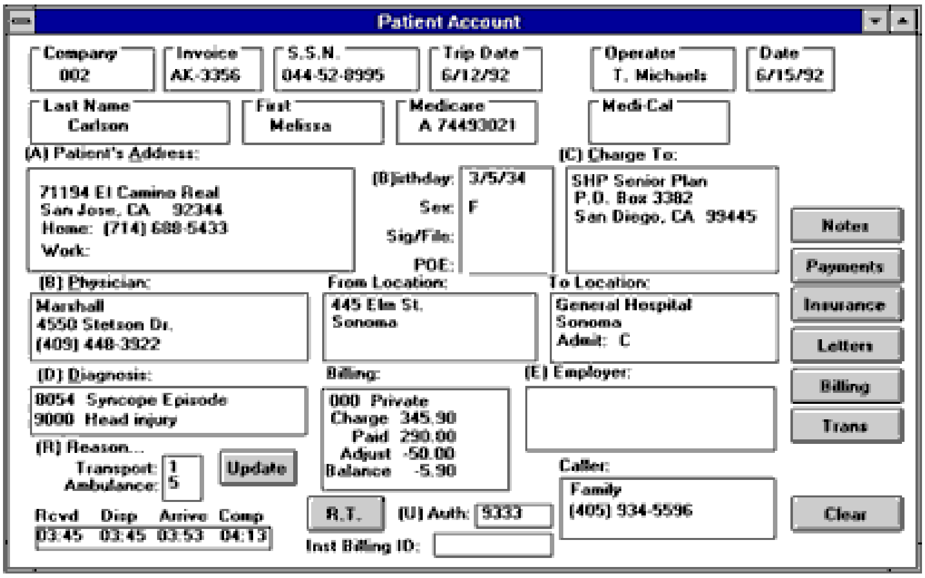
If you don't save, changes from the last 4 seconds will be permanently lost.

- Try to use modal dialogs when a finite task exists, for tasks with no fixed duration, **modeless** dialogs are normally the preferable choice
- Try to keep the user working in no more than three modeless windows at any one time.
- **Non-modal or modeless** dialog boxes are used when the requested information is not essential to continue, and so the window can be left open while work continues elsewhere.
- A type of modeless dialog box is a toolbar, items in the toolbar can be used to select certain features or functions of the application

11. Use Controls Correctly

- Controls are the visual elements that let the user interact with application.
- GUI designers are faced with many controls to choose.
- Each new control brings with it expected behaviors and characteristics.
- Choosing the appropriate control for each user task, achieves:
    - higher productivity
    - lower error rates
    - higher overall user satisfaction.

Bad design



Company: 002
Invoice: AK-3356
S.S.N.: 044-52-8995
Trip Date: 6/12/92
Operator: T. Michaels
Date: 6/15/92

Last Name: Carlson
First: Melissa
Medicare: A 74493021
Medi-Cal:

(A) Patient's Address:
71194 El Camino Real
San Jose, CA 92344
Home: (714) 688-5433
Work:

(B)irthday: 3/5/34
Sex: F
Sig/File:
POE:

(C) Charge To:
SHP Senior Plan
P.O. Box 3382
San Diego, CA 99445

(B) Physician:
Marshall
4550 Stetson Dr.
(409) 448-3922

From Location:
445 Elm St.
Sonoma

To Location:
General Hospital
Sonoma
Admit: C

(D) Diagnosis:
8054 Syncope Episode
9000 Head injury

Billing:
000 Private
Charge 345.90
Paid 290.00
Adjust -50.00
Balance -5.90

(E) Employer:

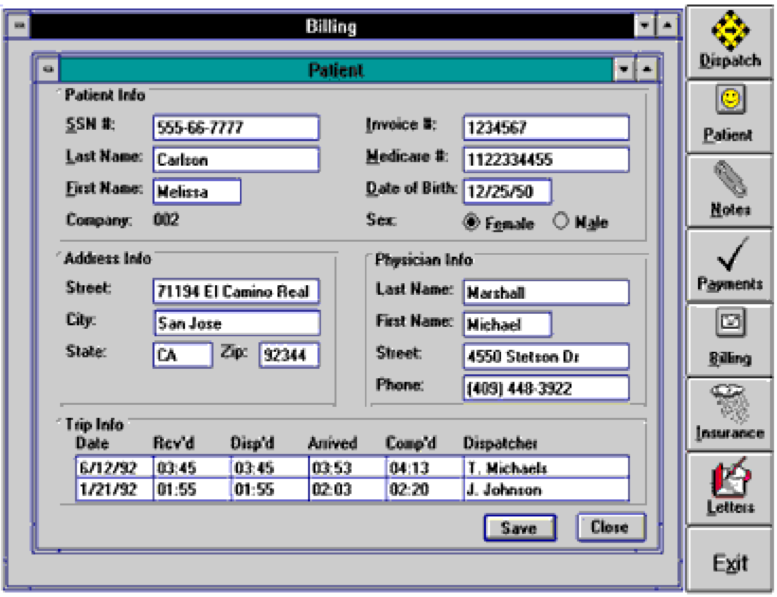
(R) Reason...
Transport: 1
Ambulance: 5

Caller:
Family
(405) 934-5596

Rcvd	Disp	Arrive	Comp
03:45	03:45	03:53	04:13

(U) Auth: 9333
Inst Billing ID:

Good design



Patient Info
SSN #: 555-66-7777
Invoice #: 1234567
Last Name: Carlson
Medicare #: 1122334455
First Name: Melissa
Date of Birth: 12/25/50
Company: 002
Sex: ● Female ○ Male

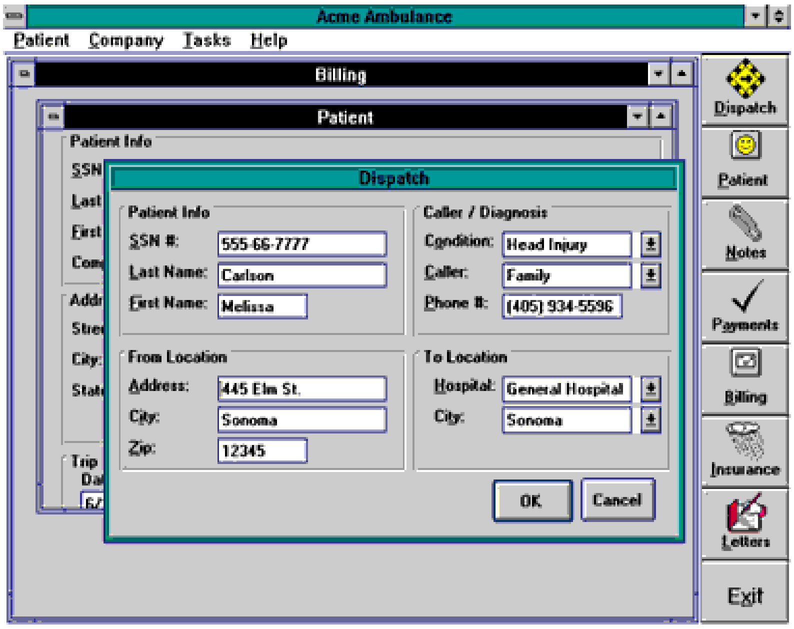
Address Info
Street: 71194 El Camino Real
City: San Jose
State: CA Zip: 92344

Physician Info
Last Name: Marshall
First Name: Michael
Street: 4550 Stetson Dr
Phone: (409) 448-3922

Trip Info

Date	Rcv'd	Disp'd	Arrived	Comp'd	Dispatcher
6/12/92	03:45	03:45	03:53	04:13	T. Michaels
1/21/92	01:55	01:55	02:03	02:20	J. Johnson

More good design



Patient Info
SSN #: 555-66-7777
Last Name: Carlson
First Name: Melissa

Caller / Diagnosis
Condition: Head Injury
Caller: Family
Phone #: (405) 934-5596

From Location
Address: 445 Elm St.
City: Sonoma
Zip: 12345

To Location
Hospital: General Hospital
City: Sonoma

Lecture 4

Embedded Systems Programming

- **Embedded Systems Design Goal:**
  - **Reliability:**
    - Mission Critical
    - Life-Threatening
    - 24/7/365
    - Can't reboot!

  - **Performance:**
    - Multitasking and Scheduling to achieve high performance measures:
      - Short response time
      - High throughput (rate of processing work)
      - Low utilization of computing resource
      - High availability
      - Fast data compression and decompression
      - High bandwidth
      - Short data transmission time
    - Optimized I/O ➔ Assembly Language
    - Limits, Inaccuracies of Fixed Precision

  - **Cost:**
    - Consumer Market: minimize manufacturing cost.
    - Fast Time to Market Required
    - No chance for future modification.

- **Real-Time System**
  - Reactive embedded system.
  - Real-time systems process events which occurring on external inputs cause other events to occur as outputs.
  - Minimizing response time is usually a primary objective, or otherwise the entire system may fail to operate properly.
  - This means parallel processing is must.
  - **Soft** Real-Time System: Compute output response as fast as possible, but no specific deadlines that must be met.(e.g. Online Databases)
  - **Hard** Real-Time System: Output response must be computed by specified deadline or system fails. (e.g. Flight Control System)
  - Real-time systems need speeding up technologies (OpenMP and MPI) considered as a support programming tool ensuring high performance added to the Embedded C as a major programming tool.

- **Multi-Tasking and Concurrency:**
  - Most real-time systems are also embedded systems with several inputs and outputs and multiple events occurring independently.
  - Separating tasks simplifies programming, but requires switching back and forth among the different threads of computation (multi-tasking).
  - Concurrency is the appearance of simultaneous execution of multiple tasks. So, concurrency is an important issue for real time embedded systems via the usage of multithreads.
  - **Ex:** Three Concurrent Tasks Within a Programmable Thermostat

```
/* Monitor Temperature */
do forever {
        measure temp ;
        if (temp < setting)
                start furnace ;
        else if (temp >
                    setting + delta)
                stop furnace ;
}
```

```
/* Monitor Time of Day */
do forever {
        measure time ;
        if (6:00am)
                setting = 72°F ;
        else if (11:00pm)
                setting = 60°F ;
}
```

```
/* Monitor Keypad */
do forever {
        check keypad ;
        if (raise temp)
                setting++ ;
        else if (lower temp)
                setting-- ;
}
```

Save energy & money using your programmable thermostat to adjust your home to a user-set temperature program

- <u>**Arduino Simple Program Example**</u>

```
// Example 01 : Blinking LED
const int LED = 13; // define an integer constant with name LED and value 13
void setup ( )
{
  pinMode(LED, OUTPUT);  // define the functionality of the digital pin 13 as an output
}
void loop()
{
  digitalWrite(LED, HIGH);  // operate the LED
  delay(1000); // wait 1000 m sec =  1 sec
  digitalWrite(LED, LOW);   //shutdown the LED
  delay(1000); // wait 1000 m sec =  1 sec
}
```

Lecture 5

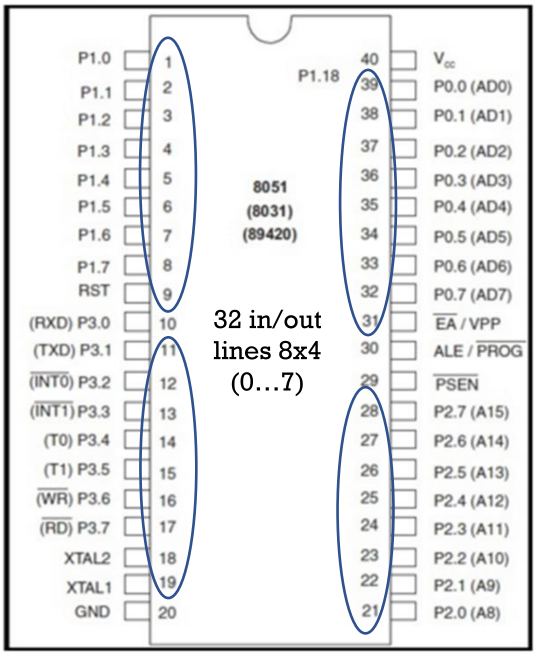
- ○ Embedded C is high level assembly language

- **<u>Embedded C Program General Format:</u>**

```
void main ()
{
    //initialize
   while (condition)
   {
       //keep doing this
   }
}
```

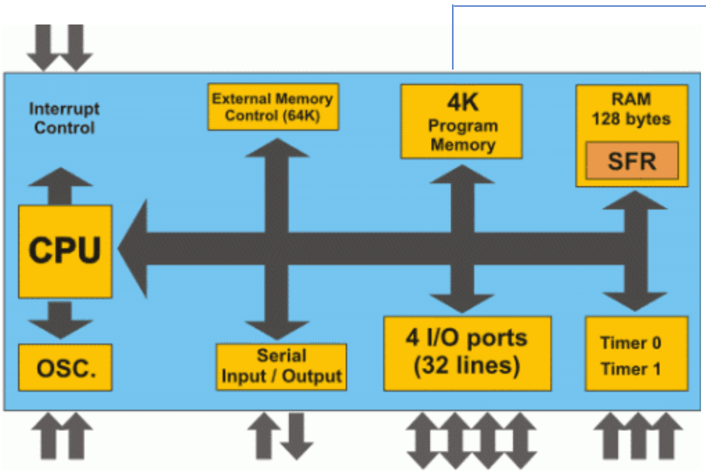
- **<u>The 8051 family Microcontroller:</u>**

Pin Diagram of 8051



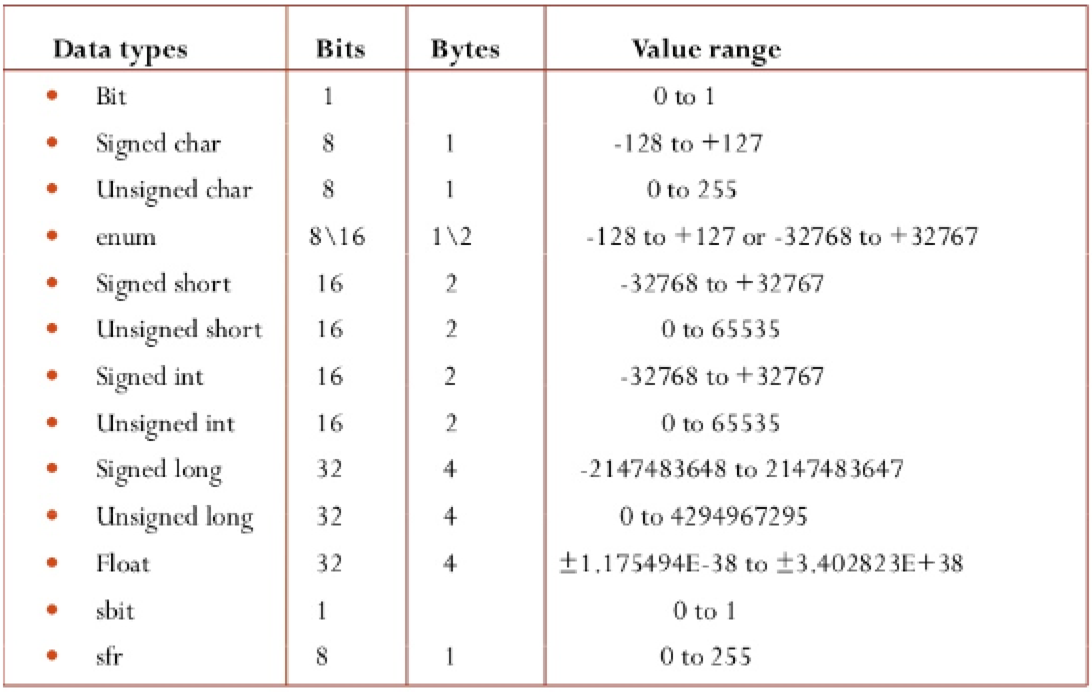
- Vcc – 5V supply
- Vss – Ground
- XTAL2/XTAL1 – Oscillator Input
- Port0 (Pins 32-39) – AD0/AD7 and P0.0 to P0.7
- Port1 (Pins 1 to 8) – P1.0 to P1.7
- Port2 – (Pins 21 to 28) – P2.0 to P2.7 and A8 to A15
- Port3 – (Pins 10 to 17) – P3.0 to P3.7
- RST – Restart 8051
- ALE – Address Latch Enable
- PSEN – Program Store Enable
- P3.0 – is used as RXD pin for serial communication.
- P3.1 – TXD pin for Serial Transmit Data.
- P3.2 – External Interrupt0 Pin, INT0
- P3.3 – External Interrupt1 Pin, INT1
- P3.4 – T0 – Clock Input for counter0.
- P3.5 - T1 – Clock Input for counter1.
- P3.6 – WR – Signal to write to the external memory.
- P3.7 – RD – Signal to read from the external memory.

- ○ **Block Diagram**



Your embedded C program will be stored here in up to:
4 x 1024 x 8 = 32768 bits

- **Embedded C Program Data types:**



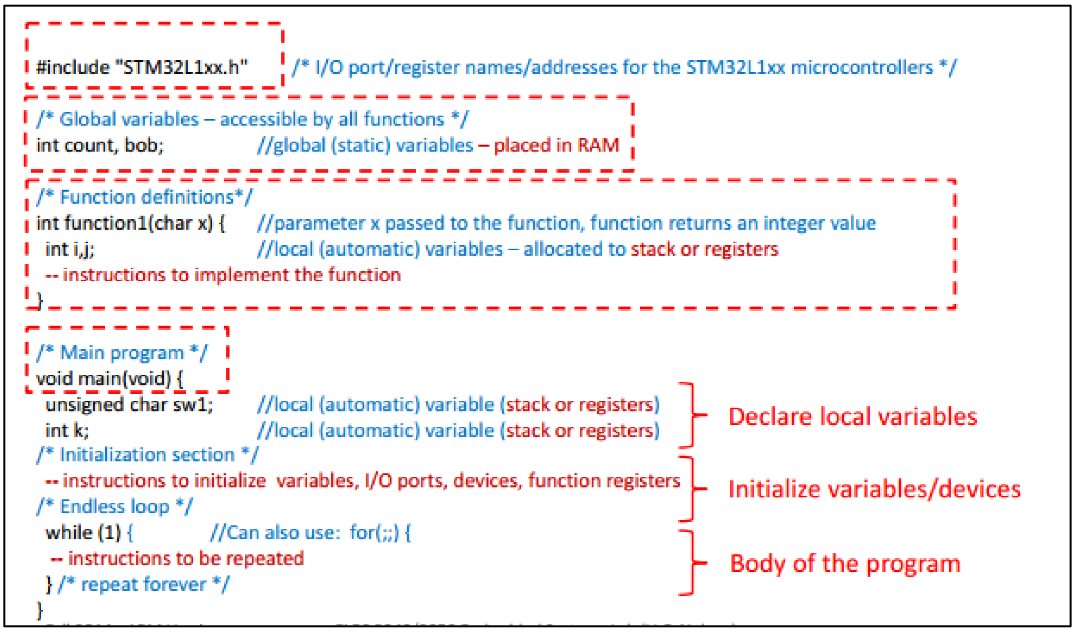
Data types	Bits	Bytes	Value range
• Bit	1		0 to 1
• Signed char	8	1	-128 to +127
• Unsigned char	8	1	0 to 255
• enum	8\16	1\2	-128 to +127 or -32768 to +32767
• Signed short	16	2	-32768 to +32767
• Unsigned short	16	2	0 to 65535
• Signed int	16	2	-32768 to +32767
• Unsigned int	16	2	0 to 65535
• Signed long	32	4	-2147483648 to 2147483647
• Unsigned long	32	4	0 to 4294967295
• Float	32	4	±1.175494E-38 to ±3.402823E+38
• sbit	1		0 to 1
• sfr	8	1	0 to 255

- **The "super loop" software architecture**
  - **Problem:** What is the minimum software environment you need to create an embedded C program?
  - **Solution:**

```
void main(void)
 {
    /* Prepare for task X */
    X_Init ();
    While (1)              /* 'forever' (Super Loop) */
     {
        X ();
     }
 }
```

  - **'super loop', or 'endless loop'**, required as there's no operating system to return, application will keep looping until the system power is removed.

- ## Basic Embedded C Program Structure



```
#include "STM32L1xx.h"     /* I/O port/register names/addresses for the STM32L1xx microcontrollers */

/* Global variables – accessible by all functions */
int count, bob;             //global (static) variables – placed in RAM

/* Function definitions*/
int function1(char x) {     //parameter x passed to the function, function returns an integer value
  int i,j;                  //local (automatic) variables – allocated to stack or registers
  -- instructions to implement the function
}

/* Main program */
void main(void) {
  unsigned char sw1;        //local (automatic) variable (stack or registers)
  int k;                    //local (automatic) variable (stack or registers)
/* Initialization section */
  -- instructions to initialize  variables, I/O ports, devices, function registers
/* Endless loop */
  while (1) {          //Can also use:  for(;;) {
   -- instructions to be repeated
  } /* repeat forever */
}
```

- ## Make a binary counter on Port 0

```
#include <reg51.h>
void main ()
{
 unsigned int i;        /*will be used for a delay loop */
 P0 = 0;                /* First, Port 0 will be initialized to zero */
 while (1)              /* super loop */
   {
     for (i = 0; i < 60000; i++) {;}        /* For delay */
     P0 = P0 + 1;    /* Increment Port 0 */
   }
}
```

- ## Central Heating Controller Partial Code

```
void main(void)
 {
 C_HEAT_Init();                  /* Init the system */
 while(1)                        /* 'forever (Super Loop) */
  {
   /* Find out what temperature the user requires (via the user interface) */
   C_HEAT_Get_Required_Temperature();
   /* Find out what current room temperature (via temperature sensor) */
   C_HEAT_Get_Actual_Temperature();
   /* Adjust the gas burner, as required */
   C_HEAT_Control_Boiler();
  }
 }
```

- **<u>Special Function Registers: SRF</u>**

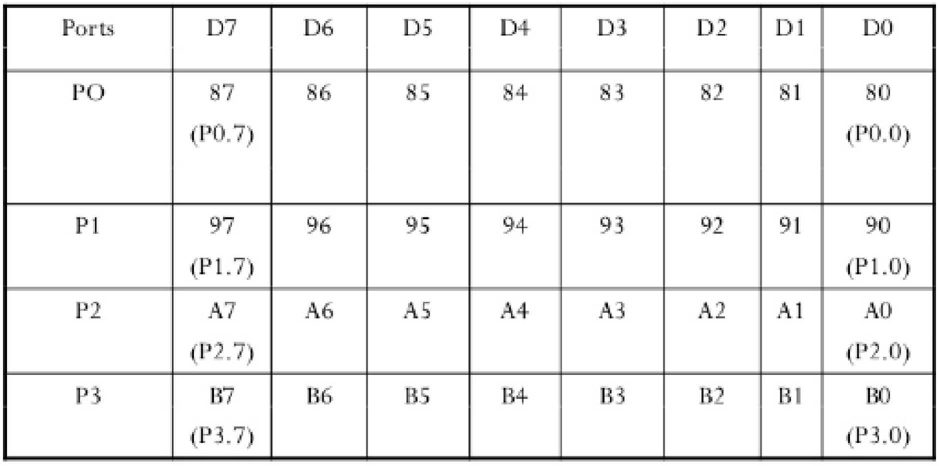
  ACC: Accumulator B: B Register
  SP: Stack Pointer DPTR(DPH,DPL): Data Pointer

  P0: Port 0 P1: Port 1 P2: Port 2 P3: Port 3

  TH0: Timer/Counter 0 High Byte
  TL0: Timer/Counter 0 Low Byte
  TH1: Timer/Counter 1 High Byte
  TL1: Timer/Counter 1 Low Byte

  SBUF: Serial Data Buffer

- **<u>Special Function Registers (SFRs) and Ports</u>**
  - Each of the four ports P0, P1, P2, P3 has 8 pins, making them 8-bit ports.
  - P0 is at address 0x80
  - P1 at address 0x90
  - P2 at address 0xA0
  - P3 at address 0xB0
  - Accessing addresses using "sfr" data type:



Ports	D7	D6	D5	D4	D3	D2	D1	D0
PO	87 (P0.7)	86	85	84	83	82	81	80 (P0.0)
P1	97 (P1.7)	96	95	94	93	92	91	90 (P1.0)
P2	A7 (P2.7)	A6	A5	A4	A3	A2	A1	A0 (P2.0)
P3	B7 (P3.7)	B6	B5	B4	B3	B2	B1	B0 (P3.0)

- **<u>Reading from (and Writing to) port pins</u>**
  - **Problem:** How do you write software to read from and /or write to the ports on an (8051) microcontroller?
  - **Background:** The Standard 8051s have four 8-bit ports, All the ports <u>are bidirectional</u>: that is, they may be used for both input and output.
  - A typical SFR header file for an 8051-family device will contain the lines:

    sfr P0 = 0x80; sfr P1 = 0x90; sfr P2 = 0xA0; sfr P3 = 0xB0;

    Having declared the SFR variables, we can read/write to the ports in a straightforward manner using a temporary buffer

- **Read from Port 1 as follows:**

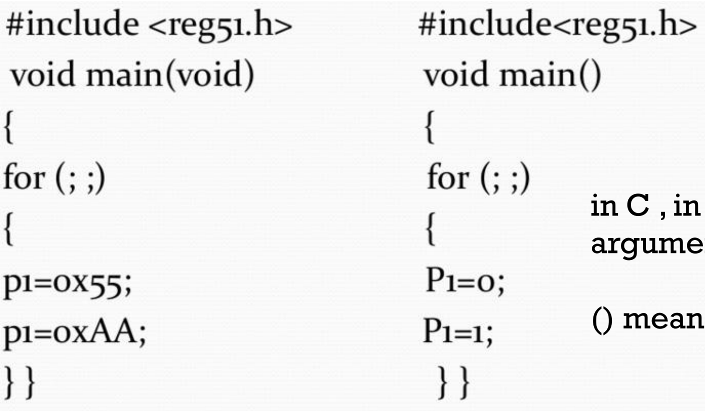
  results of reading will be saved in temporary buffer “Port_data”

```
unsigned char Port_data;
P1 = 0xFF;              /* Set the port P1 to ‘read mode’ as an input */
Port_data = P1;         /* Read from the port P1 into a temporary buffer Port_data */
```

- **Write to Port 1 as follows:**

```
unsigned char Port_data;
Port_data = 0x0F;   /* set Port_data to be temporary buffer containing binary values
                       00001111 */
P1 = Port_data;     /* Write 00001111 to Port 1 */
```

- **Toggle all the bits of P1 continuously (without delay)**

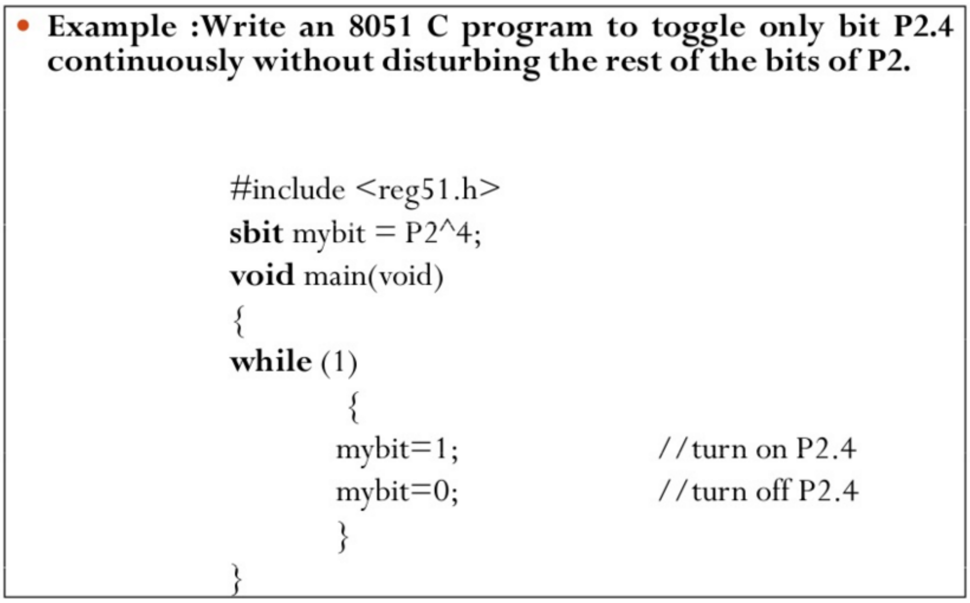


```
#include <reg51.h>          #include<reg51.h>
 void main(void)             void main()
{                            {
for (; ;)                    for (; ;)
{                            {
p1=0x55;                     P1=0;
p1=0xAA;                     P1=1;
}}                            }}
```

in C , in general ,(void) means no arguments required in function call,

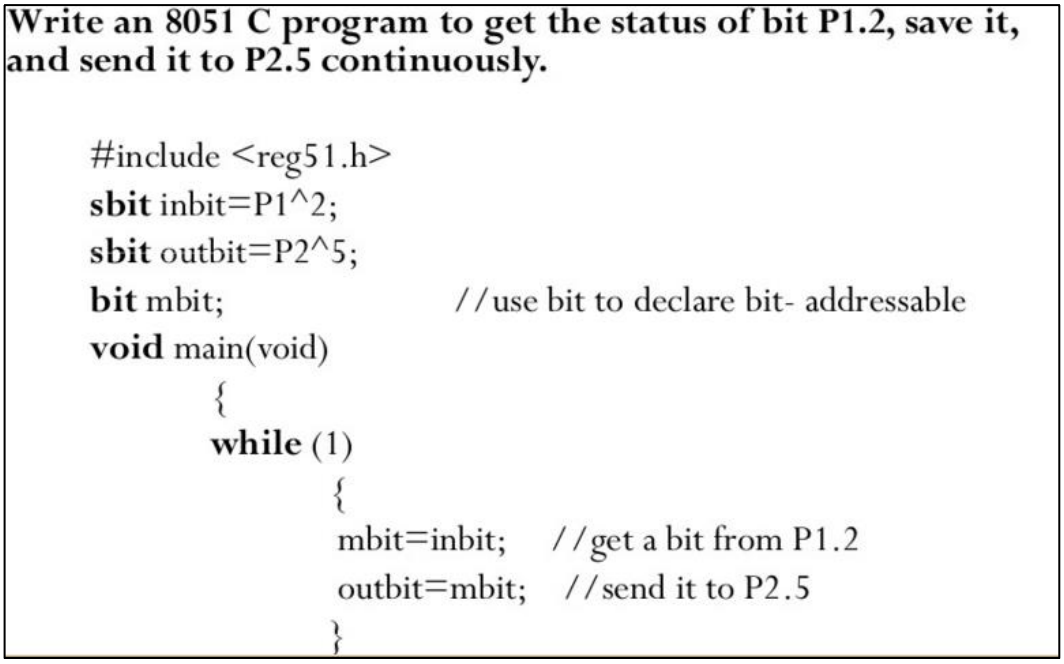
() means unspecified number of arguments

- **Other examples:**



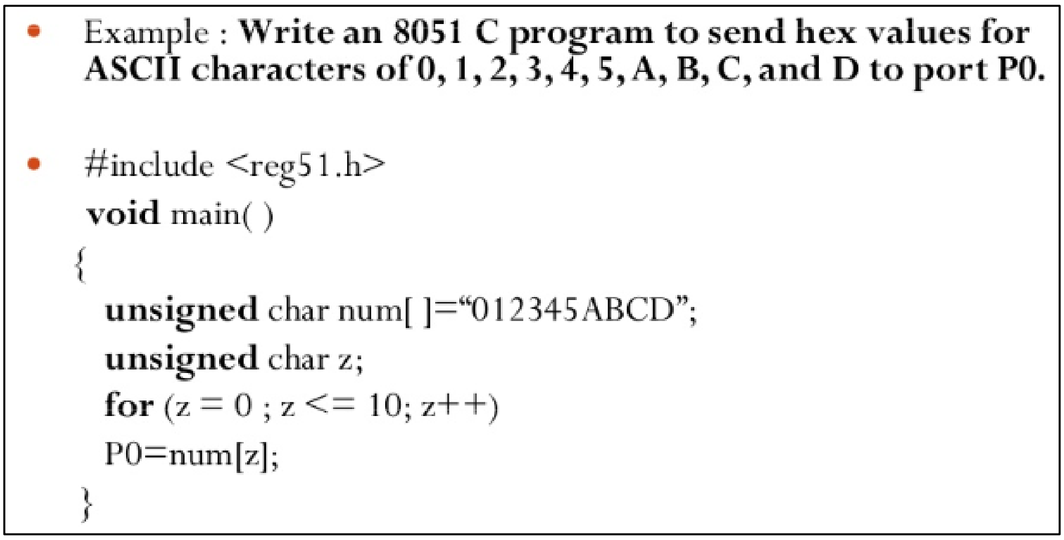
- Example :Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

```
#include <reg51.h>
sbit mybit = P2^4;
void main(void)
{
while (1)
        {
        mybit=1;            //turn on P2.4
        mybit=0;            //turn off P2.4
        }
}
```



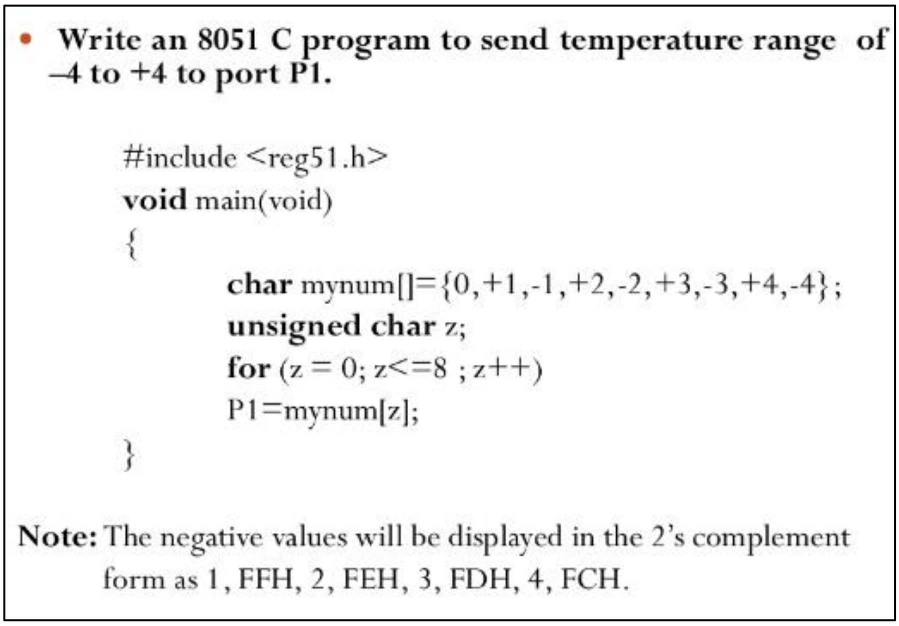
Write an 8051 C program to get the status of bit P1.2, save it, and send it to P2.5 continuously.

```
#include <reg51.h>
sbit inbit=P1^2;
sbit outbit=P2^5;
bit mbit;                    //use bit to declare bit- addressable
void main(void)
        {
        while (1)
                {
                mbit=inbit;    //get a bit from P1.2
                outbit=mbit;   //send it to P2.5
                }
```



- Example : **Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P0.**
-

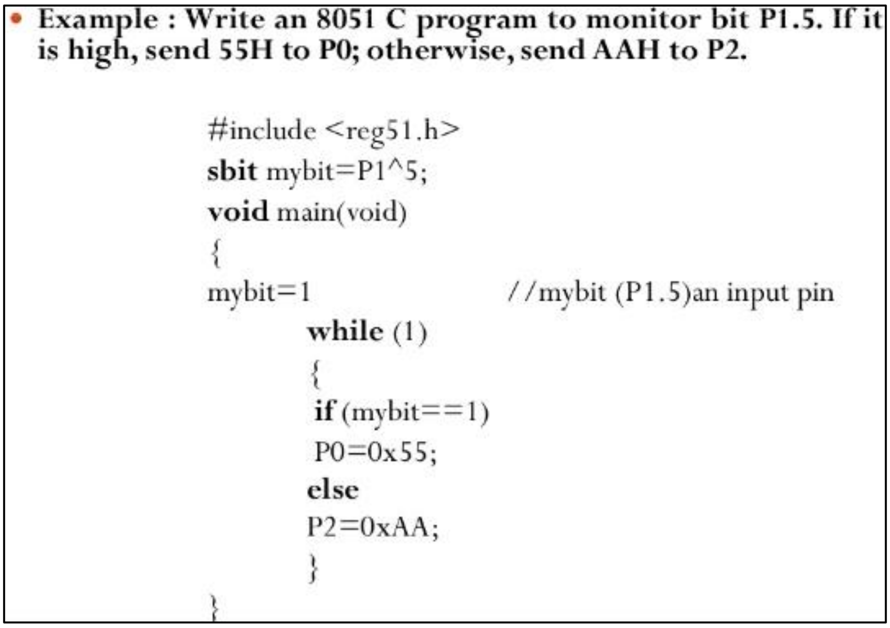
```
#include <reg51.h>
void main( )
{
 unsigned char num[ ]=“012345ABCD”;
 unsigned char z;
 for (z = 0 ; z <= 10; z++)
 P0=num[z];
}
```



- **Write an 8051 C program to send temperature range of –4 to +4 to port P1.**

```
#include <reg51.h>
void main(void)
{
        char mynum[]={0,+1,-1,+2,-2,+3,-3,+4,-4};
        unsigned char z;
        for (z = 0; z<=8 ; z++)
        P1=mynum[z];
}
```

Note: The negative values will be displayed in the 2’s complement form as 1, FFH, 2, FEH, 3, FDH, 4, FCH.



- Example : Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.

```
#include <reg51.h>
sbit mybit=P1^5;
void main(void)
{
mybit=1                     //mybit (P1.5)an input pin
        while (1)
        {
         if (mybit==1)
         P0=0x55;
        else
        P2=0xAA;
        }
}
```

- **Creating "software delays"**

```
Loop_Delay()
{
  unsigned int x, y;
  for (x=0; x <= 65535; x++)
   {
      y++;
    }
}
```

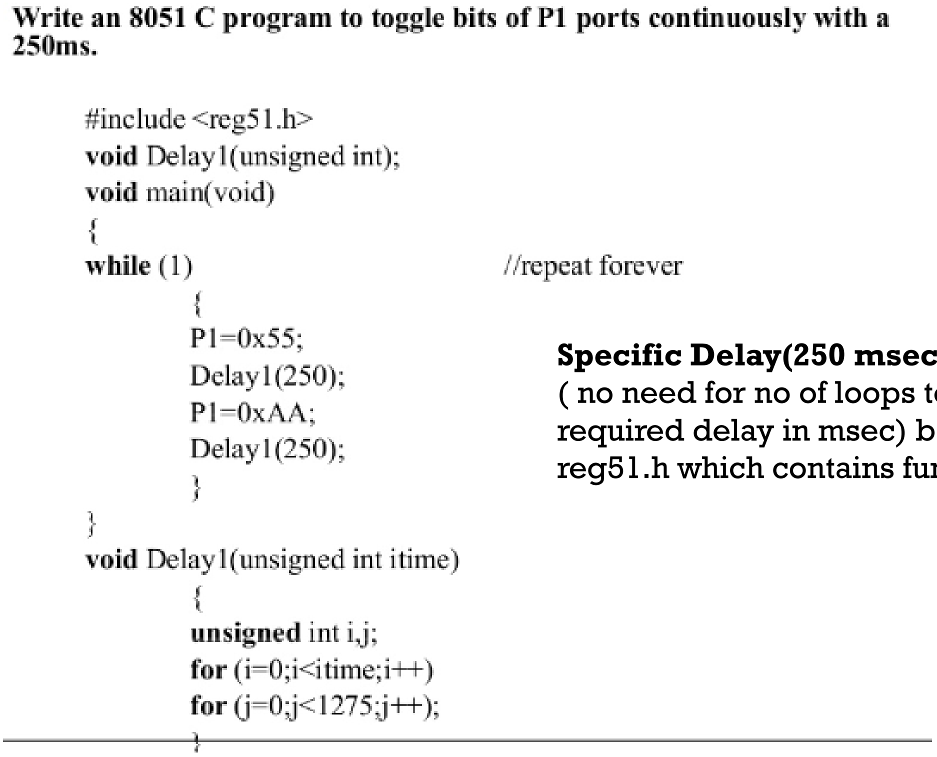
```
Longer_Loop_Delay()
{
    unsigned int x, y, z;
    for (x=0; x<=65535; x++)
     {
        for (y=0; y<=65535; y++)
         {
            z++;
         }
     }
}
```

- **Examples:**

  - Write an 8051 C program to toggle bits of P1 continuously with some delay.

```
#include <reg51.h>
void main(void)
{
unsigned int x;
for (;;)
{
     P1=0x55;
    for (x=0;x<20000;x++);          //delay size unknown
     P1=0xAA;
     for (x=0;x<20000;x++);
}
}
```

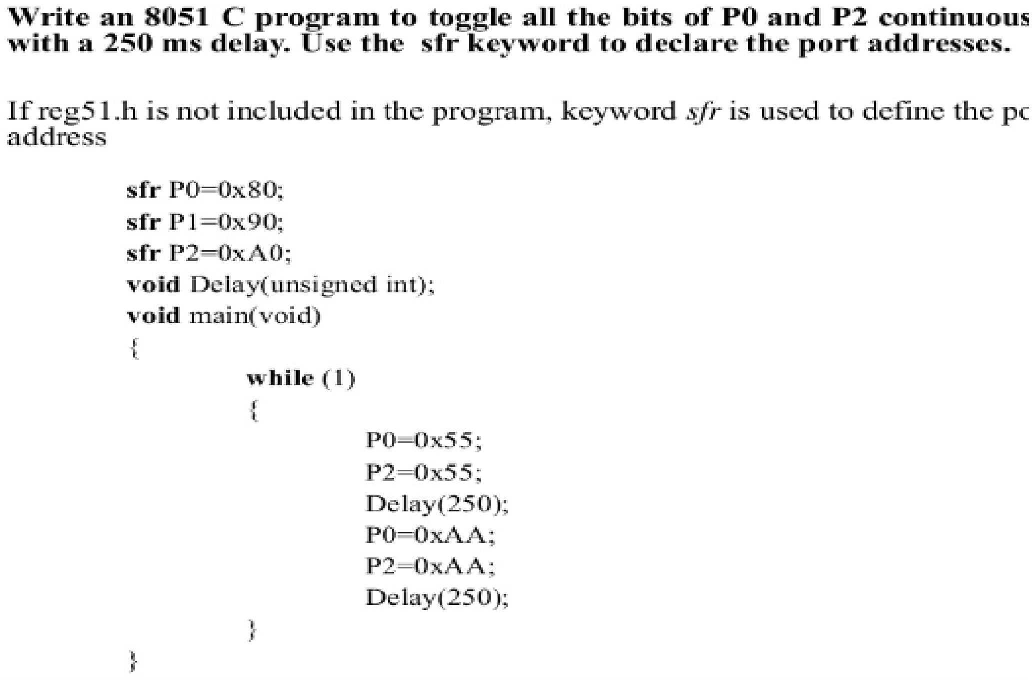
Unspecific Delay
(needs no of loops to be defined)



Write an 8051 C program to toggle bits of P1 ports continuously with a 250ms.

```
#include <reg51.h>
void Delay1(unsigned int);
void main(void)
{
while (1)                               //repeat forever
        {
        P1=0x55;
        Delay1(250);
        P1=0xAA;
        Delay1(250);
        }
}
void Delay1(unsigned int itime)
        {
        unsigned int i,j;
        for (i=0;i<itime;i++)
        for (j=0;j<1275;j++);
        }
```

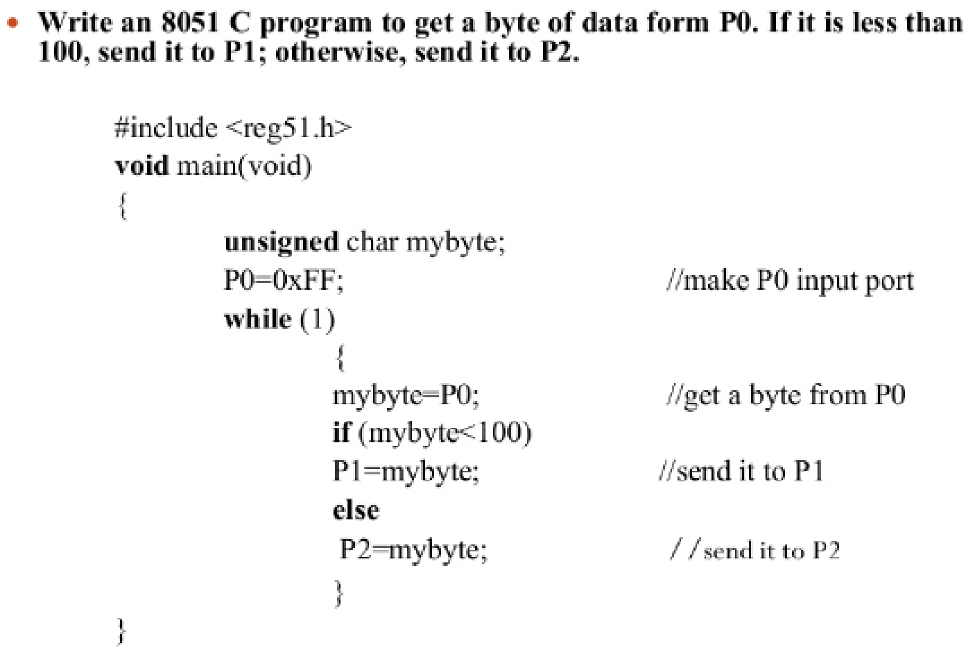
Specific Delay(250 msec)
(no need for no of loops to be defined, only the required delay in msec) based on recalling of reg51.h which contains function: Delay 1



Write an 8051 C program to toggle all the bits of P0 and P2 continuous with a 250 ms delay. Use the sfr keyword to declare the port addresses.

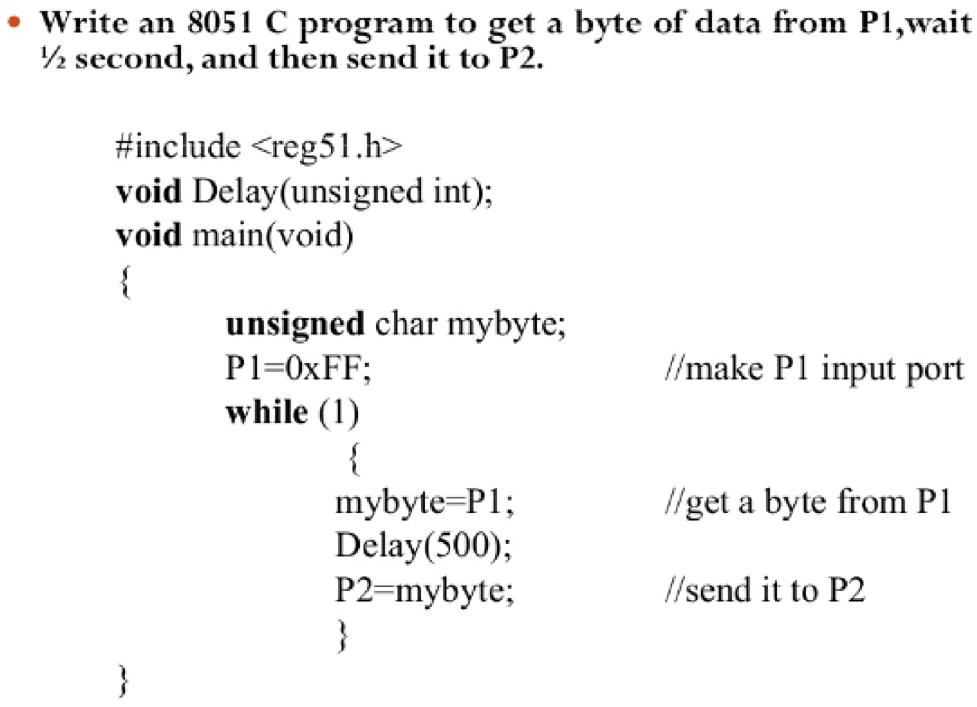
If reg51.h is not included in the program, keyword *sfr* is used to define the pc address

```
sfr P0=0x80;
sfr P1=0x90;
sfr P2=0xA0;
void Delay(unsigned int);
void main(void)
{
        while (1)
        {
                P0=0x55;
                P2=0x55;
                Delay(250);
                P0=0xAA;
                P2=0xAA;
                Delay(250);
        }
}
```



- **Write an 8051 C program to get a byte of data form P0. If it is less than 100, send it to P1; otherwise, send it to P2.**

```
#include <reg51.h>
void main(void)
{
        unsigned char mybyte;
        P0=0xFF;                    //make P0 input port
        while (1)
                {
                mybyte=P0;          //get a byte from P0
                if (mybyte<100)
                P1=mybyte;          //send it to P1
                else
                 P2=mybyte;         //send it to P2
                }
}
```



- **Write an 8051 C program to get a byte of data from P1,wait ½ second, and then send it to P2.**

```
#include <reg51.h>
void Delay(unsigned int);
void main(void)
{
        unsigned char mybyte;
        P1=0xFF;                    //make P1 input port
        while (1)
                {
                mybyte=P1;          //get a byte from P1
                Delay(500);
                P2=mybyte;          //send it to P2
                }
}
```