19CSE100 Problem Solving and Algorithmic Thinking

Modularization

Objectives

- To understand how algorithms are modularized.
- To learn how smaller units can be made flexible through the use of formal parameters.
- To understand the characteristics of well-defined modules.

Recipe book for Pizza

- Let us consider a short recipe book for Pizza.
- It might start with a contents page, that list all the recipes inside.
- It then may have a series of recipes, each having a name, followed by a set of instructions, to make that recipe.
- Suppose you wish to make *Fiorentina Pizza*, you would check where to find it in the contents, and then follow its instructions.
- In doing so you might, part way through be referred to another recipe, for example, *to make the pizza dough*.

Making of Pizza

1. *Pizza Dough*
    1. Mix yeast and water, add flour and stir.
    2. Knead for 10 minutes.
    3. Leave to rise for 30 minutes.
    4. Roll out the dough.
    5. Place in a large round pizza tin.

2. *Fiorentina Pizza*
    1. Make the pizza dough following the recipe *Pizza Dough*, leaving to rise.
    2. Spread with tomato puree, and cheese and spinach.
    3. Crack an egg into the middle.
    4. Bake in oven for 25 minutes.

3. *Tomato and Chilli Pizza*

1. Make the pizza dough following the recipe *Pizza Dough*, leaving to rise.
2. Fry the chilli pepper for 3 minutes, add onions and garlic and fry for further 5 minutes.
3. Add tomatoes and spread over the base.
4. Bake in oven for 25 minutes.

- Perhaps we came across a *pizza dough recipe* that was easier to make.

- We can change the instructions in just one place in the book and all the pizza recipes that use it now refer to the new version.

- If the dough instructions had been written out in every single recipe then we would need to make the changes in all those sets of instructions.

- Splitting up recipes (or algorithms for that matter) in this way can also make them easier to understand.

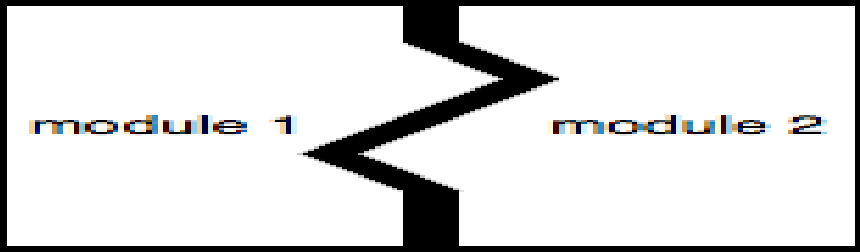
- Each separate set of instructions in a recipe book is a recipe in its own right.

✓ *It is in fact an algorithm that does something distinct and worthwhile in its own right.*

Advantages of such an approach

- This way of structuring a recipe book or algorithm has *several advantages.*
- It means that the same instructions do not need to be written out over and over again.
- This saves space, but also reduces the chances of the different versions being different.
- It also makes it easy to substitute one recipe or algorithm to do a particular thing in a different way for another.

Modularization



- Algorithms can be modularized by breaking them into *independent subprocesses*.
- A *module* is a named subprocess.
- It is an individual algorithm that should do something coherent in its own right.
  - ✓ *You do not just take a random set of instructions from an algorithm and turn them into a separate module.*
- *Modularization* is a vital element of programming that allows us to define new computable actions by assigning a name to some computable process.

✓ *Modules are vital when writing large algorithms that consist of many individual parts.*

Syntactic pattern

```
Module NAME() is
    ACTIONS
endmodule
```

Figure:The syntactic pattern - naming a subprocess as a module

- Here, *NAME* is the place holder for the name of the module (the subprocess).

- For eg. *Module average()*, *is understood to mean "execute the process named average."*

Grill a steak

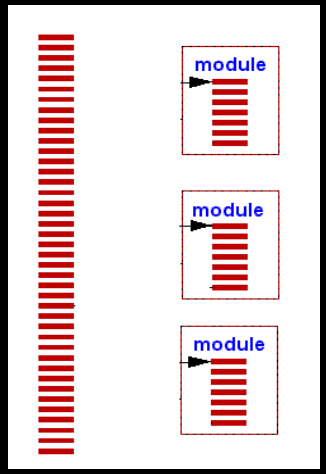

```
Module grillSteak() is
    1.  steakTemp ⟵ 75
    2.  while steakTemp < 135 do
                steakTemp ⟵ steakTemp + 13
        endwhile
endmodule
```

Figure:The *grillSteak* module

- ✓ The *grillSteak* module, shows how the steak grilling algorithm can be written as a module.
- ✓ The name *grillSteak* is associated with the process of grilling a steak.
- ✓ We can execute the subprocess by writing *grillSteak()*.

Making a dinner

- Consider, for example, a grill chef who is making dinner for a small private party.
- We might describe the entire process of making dinner as the execution of a series of modules that involve:
  - ✓baking a cake,
  - ✓fixing a salad,
  - ✓making lemonade, and
  - ✓grilling a steak.



✓ *Each of these modules can be understood as an individual, algorithemmer-defined computable action, and these actions can be invoked by simply referring to them by name.*

A *make dinner* module

```
module makeDinner() is
        bakeCake()
        fixSalad()
        makeLemonade()
        grillSteak()
endmodule
```

Figure: Making a dinner – a sequence of subprocesses

- ✓ *Since these modules have been elevated to the status of a computable action, we can also incorporate them into our flowchart notation.*

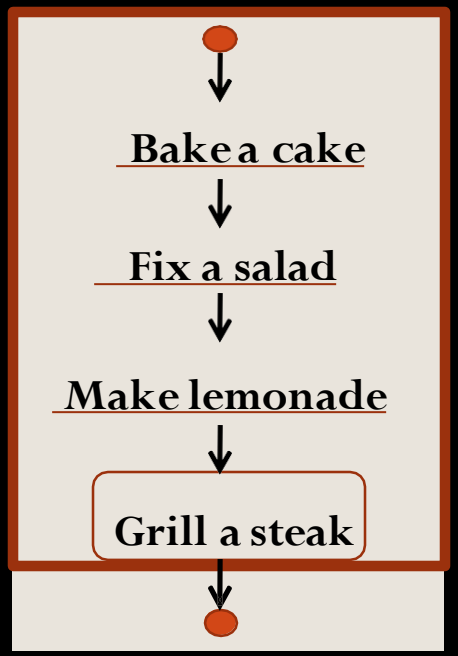


Figure: Make dinner flowchart

Well designed modules

Well-designed modules should meet several criteria. These criteria generally ensure that:

- ✓ *the module can be used in a wide variety of larger processes.*
- ✓ *the module is self-contained.*
- ✓ *any errors caused by a module have a limited effect on any system that makes use of the module.*
- ✓ *the module can be used in a wide variety of contexts.*

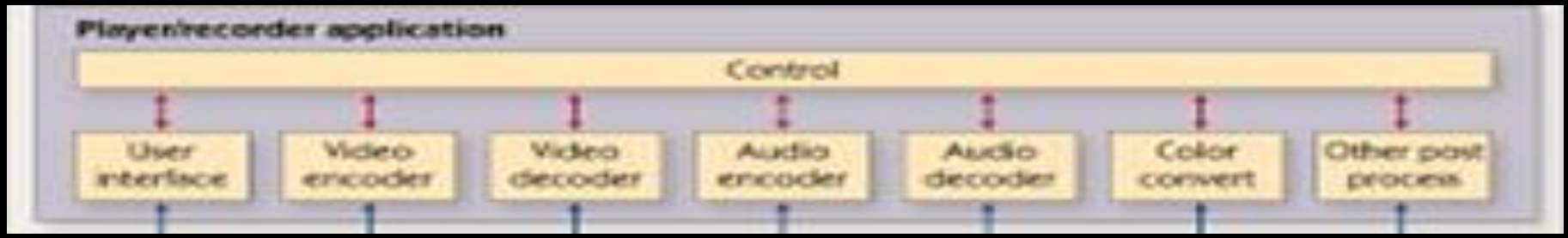
Criteria

- *Understandability:* Every module is self-contained, which implies that it can be fully understood without any knowledge of actions that take place outside of the module itself.

- *Encapsulation*: Every module affects only the data that it contains. Any errors that arise from a module are also contained within the module.

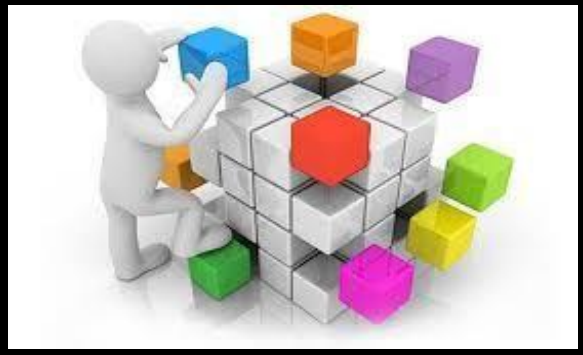
- *Composition*: Every module can be incorporated into a larger module without special treatment.

Summarizing modularity



- ✓ *Modularity is an extremely powerful technique for designing complex algorithms.*
- ✓ *Modules allow us to isolate smaller parts of a large process such that those smaller parts are much easier to understand and manage.*
- ✓ *These smaller parts are self-contained, they can be easily inserted into other complex algorithms without needing to customize them or rewrite them.*
- ✓ *Also, although we must understand precisely what a sub module does, we do not need to fully understand the details of how a sub module actually works.*

Who uses modular approach ?



- Typical industry programming projects consist of thousands of lines of code or more.
- One single large block of programming statements would be unmanageable, incomprehensible, difficult to design, write, debug, test, etc.
- *Modular Programming*: Divide-and-Conquer approach to programming.
  - Divide into sub-problems.
  - Size of modules reduced to humanly comprehensible and manageable level.
- *Analogous developments*: manufacturing a spacecraft, or a car.
  - Manufacture individual components, test separately.
  - Assemble into larger components, integrate and test.

Module flexibility

- Modules should typically be flexible enough to be used in a variety of conditions.
- Modules are made flexible *by allowing users to feed input values into the algorithm*.

✓ *The module can then respond differently depending on the input values provided by the user.*

✓ *The number of inputs that a module accepts must be defined when a module is written.*

✓ *The modules behavior then depends upon the initial value that is provided by the user.*

Parameters of a module

- The *number of inputs* that a module accepts must be defined when a module is written.
- These are known as formal parameters.
- A *formal parameter* is a variable where the initial value is bound to the values provided as input to the module.
- The initial values provided by the module user are known as *arguments* or *actual parameter*.

Formal parameters

```
Module NAME(V1,V2,….,Vn)is
   ACTIONS
endmodule
```

Figure:The syntactic pattern - formal parameters

- The figure gives an expanded syntactic pattern for defining a module.
- The formal parameters are listed within the parentheses associated with the name.
- A formal parameter is simply a variable name and a module may have as many formal parameters as convenient for the module's function.

✓ *Of course, a module may have no formal parameters, as illustrated by all of the modules we have previously described.*

One more look at the starting temperature of grill a steak algorithm ……..

- To grill a steak, the steak has to have a starting temperature.
  It is the variable *steakTemp* in the algorithm.
- While writing the process for grilling a steak,
  - It may initially seem that we have to know the value of the *steakTemp* variable.
  - ✓ *All we have to know is that there is instruction to set a starting temperature!*

✓ *Of course, if we are ever asked to grill an actual steak, we then need to obtain the starting steak temperature to actually carry out the process.*

A flexible module for *grill a steak*

- ✓ *Remember that a module simply defines what actions should be taken to grill a steak.*
- We can make our *grillSteak* module flexible by making the starting temperature an input that is controlled from the outside of the module.

```
1.   Module grillSteak(steakTemp) is
2.   while steakTemp < 135 do
       a. steakTemp ⟵ steakTemp + 13
3.   endwhile
4. endmodule
```

Figure: A flexible *grillSteak* module

Parameters of the flexible *grill a steak* module

- The rewritten *grillSteak* module now has one ***formal parameter.***
- This implies that if we ever need to use this module to grill a steak that we must supply one ***actual parameter.***
- The actual parameter becomes the starting steak temperature for the steak grilling subprocess.
- If, for example, we want to grill a steak from a starting temperature of 94 degrees because we are grilling on a scorching hot summer day from our back porch in Phoenix, Arizona, we would write *grillSteak*(94).

✓*Can you think of circumstances that might not hold in the flexible grillSteak module?*

✓*Can you to include the identified circumstances?*

Can you think of any circumstances that might not hold?

- ✓ *There are two further circumstances that might not hold.*
- ✓ *We might sometimes want to grill a rare steak and hence stop grilling when the steak reaches 130 degrees.*
- ✓ *We might also want to grill a well-done steak and not stop grilling until the steak reaches perhaps 155 degrees.*
- ✓ *Also, perhaps it is the case that we have a very thick steak on a slow-cooking grill such that the steak's temperature only increases by 2 degrees for every three minutes the steak is on the grill.*
- ✓ ***These observations imply that we can optimize the flexibility of our grillSteak module.***

✓ *How do we achieve this ?.....*

How to optimize the existing module ?

- ✓ *By adding two additional formal parameters:*
  - the target steak temperature
  - the increase in temperature for every three minutes of grilling time.

Most Flexible Grill a Steak Module

```
1. module grillSteak(steakTemp, targetTemp, increase-
   Amount) is
2.    while steakTemp < targetTemp do
3.       steakTemp ← steakTemp + increaseAmount
4.    endwhile
5. endmodule
```

Figure: An optimally flexible *grillSteak* module

How to use the revised module?

- Three actual parameters for:
  - ✓ *starting temperature*
  - ✓ *target steak temperature*
  - ✓ *the amount by which the temperature of the steak will increase with every three minutes left on the grill.*
- Each of the following shows how we can use this *grillSteak* module to achieve very different outcomes under very different circumstances.
  - ✓ *grillSteak*(65, 130, 2)
  - ✓ *grillSteak*(94, 155, 13)
  - ✓ *grillSteak*(94, 135, 5)

Modularization characteristics

- *Manageable*: Reduce problem to smaller, simpler, humanly comprehensible sub problems.
- *Divisible*: Modules can be assigned to different teams/ programmers.
  - ✓ *Enables parallel work, reducing program development time.*
  - ✓ *Facilitates programming, debugging, testing, maintenance.*
- *Portable*: Individual modules can be modified to run on other platforms.
- *Re-usable*: Modules can be re-used within a program and across programs.

Entropy of software development

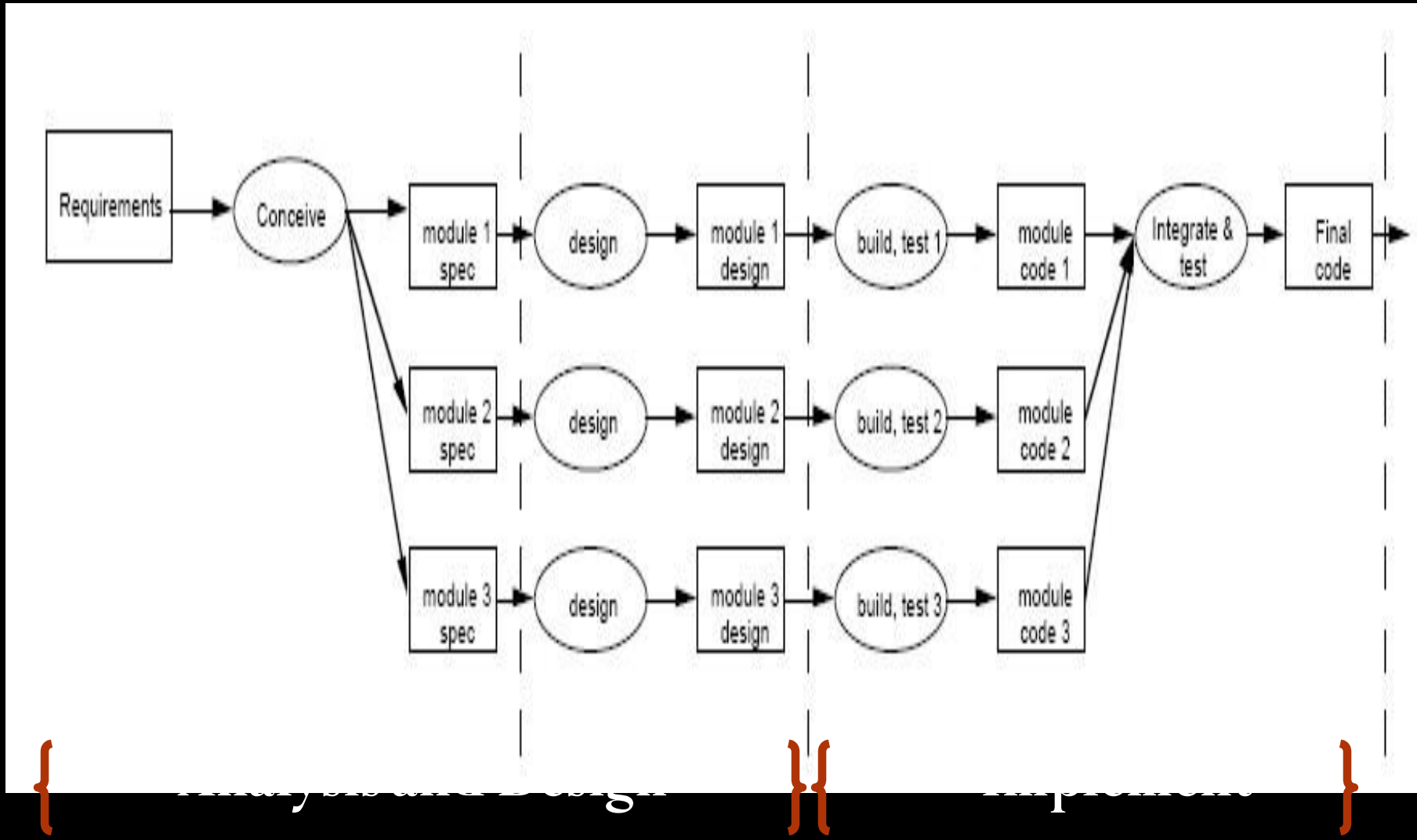
- A *monolithic algorithm is one that is a single large block of instructions.*
- *A monolithic program is one that is a single large block of programming statements.*

✓ *Where generally, fixing one bug or error always seemed to create additional new bugs or errors.*

✓ *It is one which is increasingly interconnected, unreadable, and ultimately unreliable.*

✓ *That is the entropy of software development.*

An overview ofthe development process



f this approach

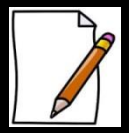
- Applications are getting more and more complicated (*growing in size and functionality*).
- It is necessary to separate them into individual pieces (whether as "*components*," "*modules*," or "*plugins*").
  - ✓ *No one is surprised anymore that operating systems are designed in a modular way.*
  - ✓ *The final product is assembled from independently developed components.*
- Modularity is a mechanism to coordinate the work of many people around the world, manage interdependencies between their parts of the project, and assemble very complex systems in a reasonably reliable way.
- It enables you to achieve and manage that complexity.

Not a silver bullet though…

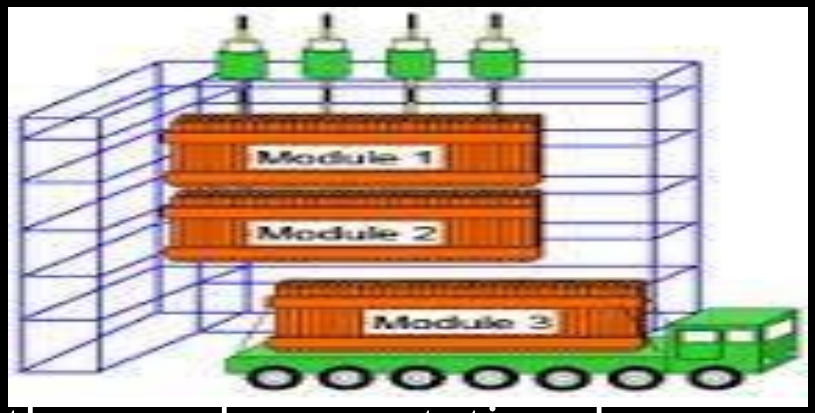
- ✓ *A print module*
  - Assume that a "*print*" module allows you to write text onto a computer screen.
  - If, for example, you wanted to write the text "*Computational Thinking*" onto the computer screen, you would express this action as *print(ComputationalThinking)*.

✓ *Draw a flowchart to print the values 0, 3, 6, 9, and 12 to the computer screen. You must use a loop.*

✓ *Write a module named "countBy3" that prints the values 0, 3, 6, 9, and 12 to the computer screen.*



What has been described?



- Modules as ways of expressing algorithms and computational processes.
- How modules allow us to break large problems into smaller units.
- How these smaller units can be made flexible through the use of formal parameters.
- The importance of modular approach to software development.

Credits

- *Computing without computers, Paul Curzon;*
- *http://www.eecs.qmul.ac.uk/~pc/research/education/puzzles/reading/*
- *web.mit.edu/16.070/www/year2001/Modular_Programming.pdf*
- *The Benefits of ModularProgramming, Copyright Sun Microsystems, 2007*
- *Google images*