Subroutines

- When a program becomes too large, it is good to decompose it into smaller chunks.
- A subroutine (or procedure) is a chunk of code which in isolation can solve a particular problem.
- While making code easier to read, a subroutine is ideal for re-use.
- This leads to 2 methods of tackling an algorithmic design:
  - Top-down approach
  - Bottom-up approach
- A subroutine is called and can call others.
- A subroutine might be called with some inputs and might give some outputs.
Recursion

• When a sub-routine calls itself, it is known as recursion.

• It is analogous to splitting the task between more people to compute.

• A recursive sub-routine needs
  – A call to itself
  – A terminating condition

• Example: finding the maximum number in a list.
Data Types

• An algorithm manipulates data, but this data can come in different flavors.
• In a programming language, the different types of data are known as data types.
• Data types vary from language to language but normally include integers and strings.
• The data type defines the operations that can be performed on the data.
• Operators are meaningless without a definition of the data type.
Variables

- Data needs to be referenced somehow.
- A variable is a name given to a specific data element
  - More specifically a variable refers to the place holder of the data
- Variables can represent input, output and also temporary data.
- A variable holds data of a specific data type.
- The storage space of a variable depends on the data type.
Complex Data Types

- Lists
- Arrays or Tables
- Queues or Stacks
- Pointers
- Trees
- Databases

Different data types and operands make most of the difference between one programming language and another.
Programming Languages

• A program is an official and formal rendition of the algorithm suitable for computer execution.

• A programming language defines the operations that can be performed on data types.

• A programming language is normally defined through the use of the Bacchus-Naur Form (BNF).
  – Ex : <while statement> :-
    
    while <condition> <statement> endwhile

    <condition> :- <num><numoperator><num>

    <num> :- 0|1|2|3|4|5|6|7|8|9|1<num>|….9<num>
Programming Languages (2)

• The BNF states the syntax of the language.
• A program also needs precise semantics
  – Each statement needs a meaning
• Semantics of programming languages need complex formal specifications and is rarely done.
• After a program is written according to syntactic and semantic rules, it is either
  – Compiled
  – Interpreted

to be understood by a computer.
• Most basic computer language that is used is assembly offering a 1:1 mapping to machine code.
Programming Models

• While the minimum set of algorithmic constructs exist in all programming languages, there are various methods to tackle the construction of an algorithm.

• Programming languages have been built around these methods (models) to facilitate their use.

• Yet the language does not enforce the correct usage of the programming model.

• Different models can lead to very different algorithms for the same problem.

• Some problems are more suitable to one model than another improving algorithm development time and correctness.
Imperative Programming

• This is the first and most widely used model.
• An algorithm is made up of a sequential list of instructions for the computer to execute.
• Languages include Basic, Pascal and C.
• We will use this model in this course.
• It is the easier to learn but results in longer development time.
Object Oriented Programming

• In the real world, we are used to dealing with objects that have an isolated existence.
• In OOP, entities (objects) are created that can be treated in isolation and re-used.
• Each object has an expected behavior and an identity.
• The internals of an object is built up using imperative programming
• Languages include Java and C++.
• OOP improves development time and correctness.
• Example: Car Parts
Declarative Programming

• In this model, we state the facts and let the system come up with conclusions depending on our question.
• We state relationships between facts using logical statements.
• It is ideal for problems which deal with facts and relationships
• Languages include Prolog and Lisp (in a way).
• Example: Family relationships
Functional Programming

- This evolved from lambda calculus, a way to represent computation mathematically.
- It uses functions and functional relationship as a model, very similar to how mathematical formulas are written.
- Ideal for algorithms which have to be formalized.
- Theorems form the foundation of the algorithm.
- It is natural to use recursion in functional programming
- Languages include ML and Miranda.
- Example: Factorial
Parallel Programming

- Algorithms are seen as tasks which execute independent of each other and concurrently.
- Tasks communicate by sending messages to each other.
- This model is used when more than one computer is involved in the process and it is advantageous or necessary to have concurrent execution.
- Languages include Occam which is based on CSP.
- Example: Dining Philosophers