Objectives

- This topic should allow students to
  - Understand the importance of abstraction in programming
  - Recognise when a function would be useful.

The plan

- Today we’ll introduce the idea of procedural abstraction and look at some examples.
- Next topic will cover the detail and get you to start using functions.

Abstraction

- In computer science abstraction means identifying and naming something complex and then using that name to refer to the complex thing.
  - Data abstraction:
    - design a data structure to hold details about a bank account; call it bank_account
  - Procedural abstraction:
    - Give a name to a piece of code that does something; use the name to get that something done.
Procedural Abstraction

- Give a name to an algorithm
- Use the name to execute that algorithm within another one.
- Example
  - procedure sqrt (x)
    {algorithm for computing sq. root}
  - sqrt (789.54)
    Means APPLY the sqrt procedure to the number 789.54.

Abstraction contd./

- Example 2
  - procedure sort (list)
    {algorithm for sorting a list}
  - sort (list-of-names)
    Means APPLY the sort procedure to the list of names.
- Abstraction allows you to separate details of how to carry out the procedure from the algorithm that uses the procedure.

Procedural Abstraction in C++

- In C++ procedural abstractions are called **functions**.
- A function encapsulates and names an algorithm to solve a particular problem.
  - That algorithm can be used in a program simply by calling the function by name, generally supplying information and receiving a result.
- sqrt, pow, setprecision are built-in functions that we have used.
- We can design and implement our own functions as we need them.

Some built-in functions

- By including `<cmath>` you can use
  - pow(i,n)
    - A built-in function to raise i to the power of n. We used it in the investment table program.
  - sqrt(x), fabs(x), sin(x), cos(x)
    - sqrt returns the square root.
    - fabs returns the absolute value of a double
- Note that you don’t have to worry about how these functions are coded.
Other functions we might have liked to have

- \texttt{ppsi(diameter, price)}
  - To compute the price per square inch of a circular pizza (or any circle).
  - (Actually this year we didn’t do the pizza price exercise.)
- \texttt{is\_a\_factor\_of(n, f)}
  - Which would return true if \((n\%f)==0\)
  - Useful for testing for primes or perfect numbers.

Other functions we might like

- \texttt{gross\_pay(hours, rate), tax(pay, tax\_credit)}
  - These could simplify a payroll program.
- \texttt{letter\_grade(grade)}
  - Could turn e.g. 45 into “P3”, 74 into “D”
- \texttt{leap\_year(year)}
  - Returns true if year is a leap year. Useful in date calculation programs.
  - We can write such functions
  - While that’s not as handy as built-in ones, user-defined functions have lots of benefits.

The role of functions

- Hide detail
  - We don’t need to care how \texttt{sqrt(x)} is implemented.
  - While we are writing the main payroll program we can ignore how \texttt{gross\_pay(hrs, rate)} operates.
- Increase readability
  - \texttt{is\_a\_factor\_of(n,d)} should be more meaningful than \((n\%d == 0)\)

Role of functions cont./

- Allow reuse of code
  - \texttt{sqrt} function is useful in all sorts of mathematical programs.
- Modular software development:
  - See next slides. Modular approach …
  - Aids program design
    - Use a function to implement subtasks of an algorithm.
    - Allows incremental development and testing
    - Allows division of labour.
Modular System Development

- Build software like you would build a car.
- Identify the components you need.
- Specify what each needs to do
- Buy one that meets the spec or get someone to build one.
- Test each component before putting it in the system.
- To improve the car, replace components with better versions.

Modular Software Development

- Large programs must be broken down into components.
- Each component is implemented by a function.
- Specify the behaviour of each component and the interface between the system and the component.
- Develop and test components separately before integrating into the whole.