Error Free Software

COSC1076
Semester 1 2019
Week 07

RMIT UNIVERSITY

Admin

- Assignment 2 Preparation
  - Register your group ***this week***
  - Group size 4
  - Groups formed in Lab classes
    - There will be updates in labs
- Week 8 Online Quiz
  - More info at end of lecture
- Friday
  - Public Holiday

Review: Linked Lists

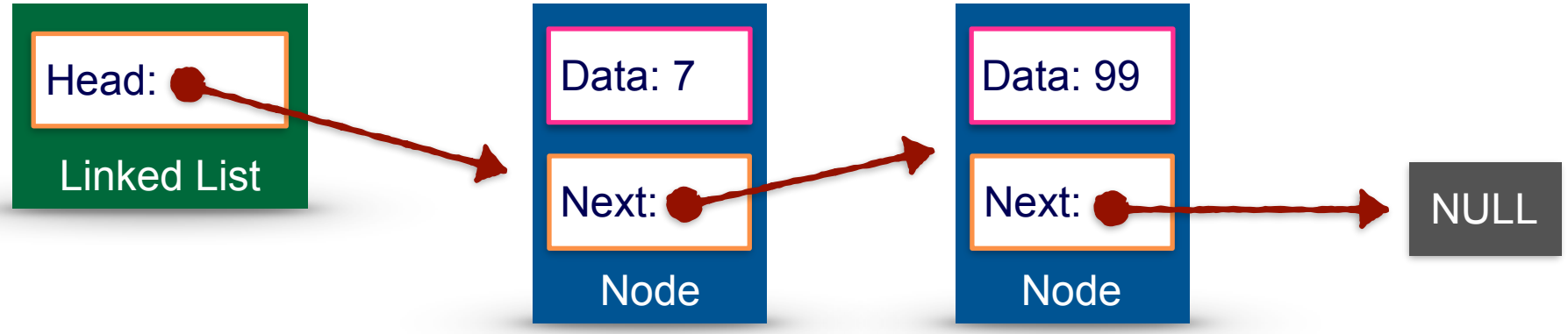
RMIT UNIVERSITY

Abstract Data Types (ADTs)

- An ***Abstract Data Type (ADT)*** consists of:
  1. An interface
     - A set of operations that can be performed.
     - Usually contained in a header file
  2. The allowable behaviours
     - The way we expect instances of the ADT to respond to operations.
- The implementation of an ADT consists of:
  1. An internal representation
     - Data stored inside the object's instance variables/members.
  2. A set of methods implementing the interface.
     - Usually contained in a code file
  3. A set of representation invariants, true initially and preserved by all methods

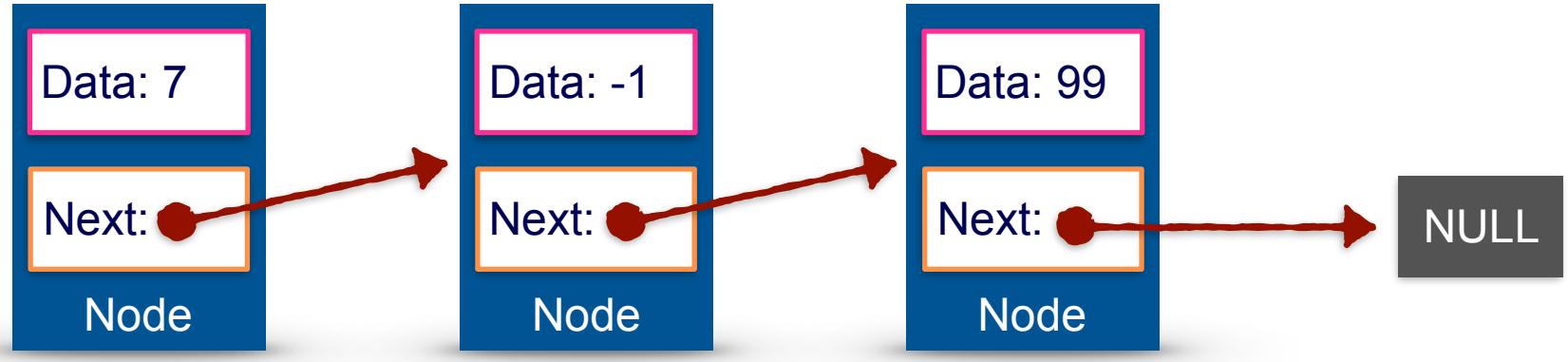
Linked List ADT

- To use the list of nodes, a ***Linked List ADT*** is created which:
  - Internally contains a pointer to the ***head*** (first node) of the list
  - Defines methods to interact with the linked list
    - These methods are similar to those used on the ParticleList!



Linked List

- A ***Linked List*** is a data types that:
  - Stores elements of the same type
  - Stores the elements as a sequential “chain”
  - The last element of the chain is “null"



Linked Lists vs Arrays/Vectors

	Array/Vector	Linked List
Element Types	All elements are same type	All elements are same type
Resizeable	Yes Requires Copy	Yes
Storage Layout	Contiguous	Non-Contiguous
Cell lookup	Constant Time Random Access	Linear Time Random Access

Linked Lists vs Arrays/Vectors

	Array/Vector	Linked List
Insert at Front	Linear Time Requires Copy	Constant Time
Insert at Back	Constant Time	Linear Time (if no back ptr) Constant Time (back ptr)
Delete at Front	Linear Time Requires Copy	Constant Time
Delete at Back	Constant Time	Linear Time (if no back ptr) Constant Time (back ptr)

Linked Lists Continued

RMIT UNIVERSITY

More operations

- Today we will inspect three additional operations:
  - get
  - contains
  - deleteFront
- Remember, before starting on any method, draw the picture first!

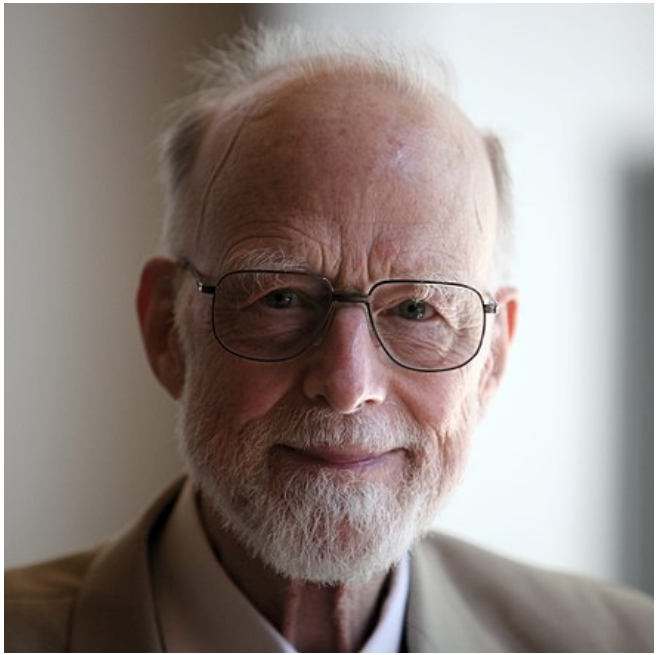
How can we ensure our code is 100% Correct?

RMIT UNIVERSITY

C.A.R. Hoare on Software Design

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

-- Sir Charles Anthony Richard Hoare



Theories of Writing Software

RMIT UNIVERSITY

Key Question

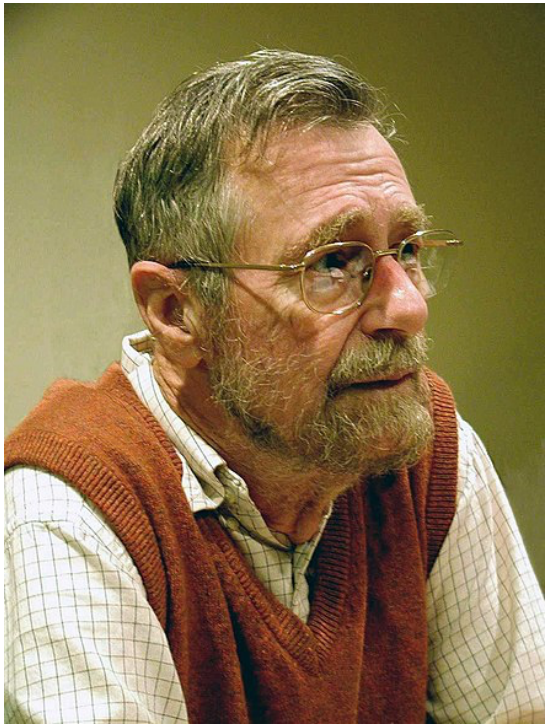
“Where do I have to look to know what my code does?"

Dijkstra on Programming

"… I have seen many programming courses that were essentially like the usual kind of driving lessons, in which one was taught how to handle a car instead of how to use a car to reach one's destination.

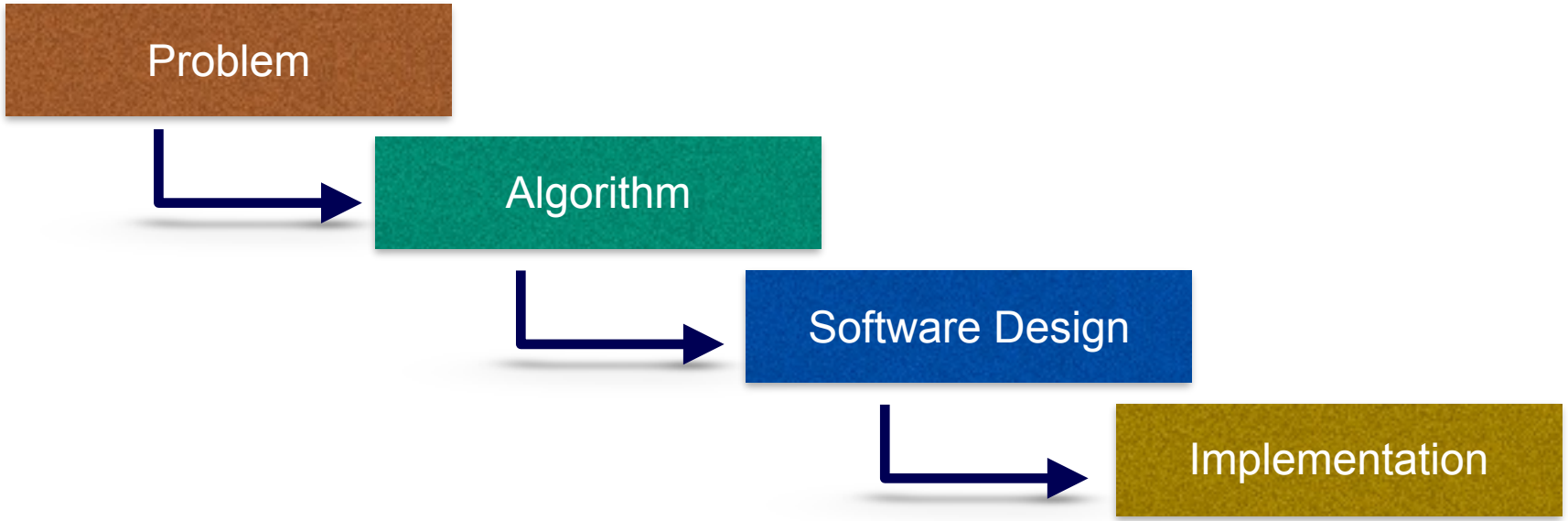
My point is that a program is never the goal itself; the purpose of a program is to evoke computations and the purpose of the computations is to establish a desired effect. Although the program is the final product made by the programmer, the possible computations … are the true matter …"

-- Edsger W. Dijkstra
(Notes on Structured Programming, 1969)



Problem Solving

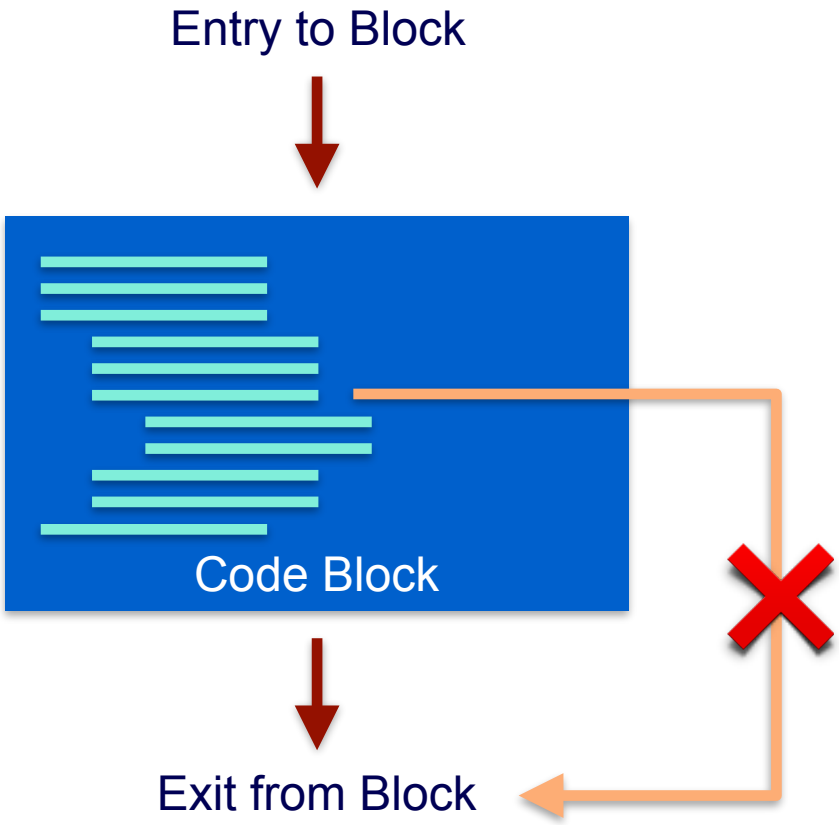
- Problem Solving is about find software solutions to problems
  - It's an obvious statement, but what does it actually mean?



Structured Programming

- ***Structured Programming*** is the concept the every block of code has a:
  - ***Single point of entry***
  - ***Single point of exit***
- Structured Programming restricts code to:
  - Sequence
  - Selection
  - Iteration (repetition, recursion)
  - Sub-routines (functions, methods, classes)
  - Blocks
- Theoretical basis:
  - Any function computable by a Turing machine can be solved using only these above structures

Structured Programming



Structured Programming

- Structured Programming allows for clear code modularity
  - A structure, function, method or block can be treated as a black-box
  - It is clear about what happens to enter the block
  - It is clear about what happens to leave the block
    - This is very useful for code provability
- Modular code blocks are an establish principle of good software design
  - These also connect to the concept of programming-by-contract

break, continue & goto | multiple returns

- Three common keywords (operators) in many programming languages defy the concept of structured programming
  - break - terminate a loop early
  - continue - stop a loop at it's current point and restart with the next iteration
  - goto - jump to any line of code and resume execution
- Multiple returns are where a function or method may exist at different points
- All of these concepts defy the single-point-of-exit paradigm

switch

- Switch statements also use the break keyword
- It is a misconception that switch statements are identical to if-elseif statements
  - This is because of “fall through” statements
  - Fall-through results in a given switch element executing for different reasons
- Therefore, this defies both single-point-of-entry and single-point-of-exit.

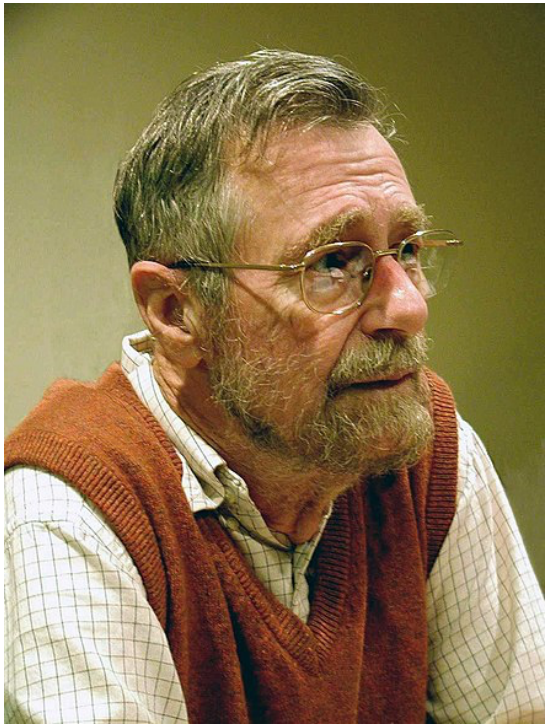
Common “arguments” for these keywords

- Code can be written more efficiently
  - Not true, the same efficiency can be achieved in structured programming
- Code is “clearer” or “easier” to read
  - In general, no, as the keywords reduce the modularity of each code block
  - That is, the programmer must inspect more code to determine what happens
- By following good principles, these can be used appropriately
  - So, by the same argument, why not just stick to structured programming?
- It is better for handling errors
  - Exception handling is even better

Dijkstra on goto

"Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later… did I become convinced that the go to statement should be abolished from all 'higher level' programming languages…"

-- Edsger W. Dijkstra
(Goto Considered Harmful, 1968)



Defensive Programming

- ***Defensive Programming*** is the concept that:
  - You must account for anything and everything that a user of your code could possible do
  - You must handle any possible input
  - You must account for any possible state of your software
- Generally, this concept is applied to:
  - Providing I/O to a program
  - Calling a Function or Method
  - Interacting with an ADT
  - Interacting with some module or block of code
- That is, you must “defend” your program from inappropriate use.

Defensive Programming

- Defensive programming frequently leads to overly verbose code
  - In general there may be more code to handle incorrect use than the actual code that performs the desired functionality
  - It may be impossible to “handle” all possible scenarios
- Defensive programming is important for:
  - Understanding the range of input & interactions with a module
  - Ensuring code is secure
  - Preventing unforeseen consequences
  - Make a module behave in a consistent and well-defined manner in all circumstances

Programming by Contract

- ***Programming by Contract*** is almost the opposite of defensive programming
  - The programmer specifies a “contract” that must be complied with
  - Both the programmer who wrote the contract and the user of the contract must comply with all aspects
  - Only things which are specified in the contract are guaranteed to happen
  - For anything which is outside of the contract, the behaviour is undefined
- Programming by Contract is general applied to:
  - Calling a Function or Method
  - Interacting with an ADT

Programming by Contract

- Generally, a contract for a function/method specifies:
  - Legal input values
  - Ranges on the input
  - Additional properties that must be true
  - Guaranteed output
  - Properties of the output
- In C++ there are no good methods** to enforce a contract
  - Generally, a contract is only specified in comments
  - ** Except:
    - Can use inheritance & templating
    - C++20 is introducing more possibilities

Programming by Contract & Defensive Programming

- Programming by Contract and Defensive Programming overlap
- For example, when implementing a method or function:
  - The contract specifies the legal range of input
  - BUT you must defensively program with respect to all legal input
- For a design aspect, it might be better to specify a broader contract simply because it is less likely for other users to make mistakes when interacting with your software

Exception Handling

- Even under a contract, there are situations in which it is permitted for something to go wrong
  - That it is known that an error can occur
    - For example, connecting to an external database
  - It is permissible for a user to “do a bad thing”
    - Attempting to read beyond the end of an array
- ***Exception Handling*** is a programming paradigm for responding to, and recovering from errors
  - Specifically, and exception occurs when the normal operation of the program has been compromised or broken

Exception Handling

- Code in executed in the try block
- If an error occurs, or exception thrown, the try block stops and execution resumes at the top of the matching catch block
  - Errors of a certain type can be caught
- Execution resumes after the try-catch regardless of which finishes

```
try {
    list->deleteFront();
    list->deleteFront();
    list->deleteFront();
} catch (std::string& error) {
    std::cerr << "Caught error: " << error << endl;
}
```

Exception Handling

- As the result of an exception
  - The normal execution of the program is halted
  - The execution is resume in the exception handle from a known point
  - The handle may attempt to “recover” the program state
  - The program continues from a consistent, known state
  - Alternatively the program quits

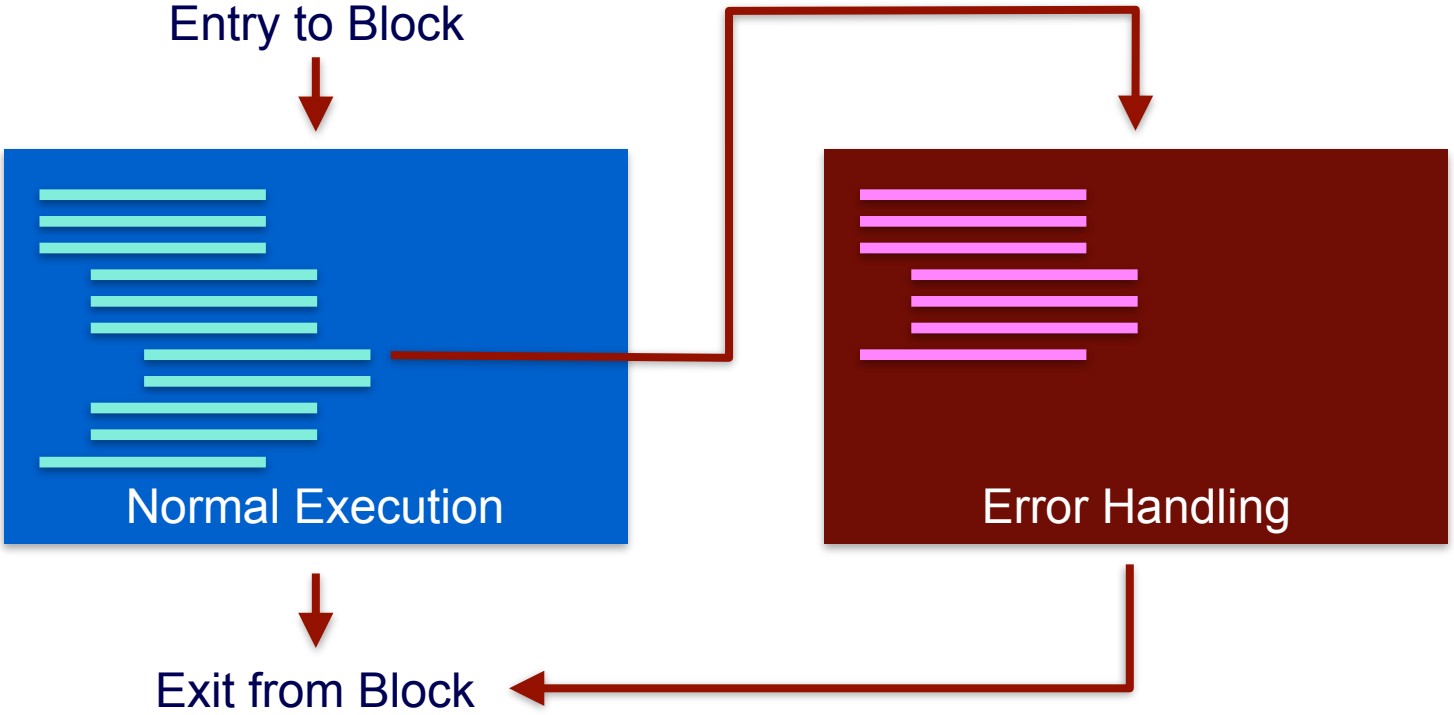
C++ Standard Exceptions

- C++ provides a number of standard exceptions that can be thrown
  - Defined in `<stdexcept>`
  - Most common exception we will use is `std::runtime_exception`
  - See: https://en.cppreference.com/w/cpp/error/exception

Exception Handling & Structured Programming

- Do exceptions violate structured programming?
  - Yes, and No
- Yes
  - Within a the flow of a single-thread execution
  - Causes multiple exit points
- No
  - Provides a more rigorous separation of normal and abnormal behaviour
  - Provides a structured programming approach to managing errors

Exception Handling & Structured Programming



Testing

RMIT UNIVERSITY

Writing Tests

- Tests are a dynamic method for determining if code is correct
  - That is the code is executed to determine it's correctness
- In principle, there should be one test for every possible "thing" that a program can do
- In general, if the state space of a program is infinite, then you would require infinite tests

Black-box Testing

- The contents of the software is unknown, that is, treated as a block box
  - You don’t have to know anything about the implementation
- Tests are written using input and output
  - Given specific input, the program is expected to produce given output
  - If the output does not exactly match, the test fails

Black-box Testing

- It is only possible to test expected behaviour
  - Testing for “errors” is only possible if there is an expected behaviour
  - The behaviour could be weak, for example, the program does not crash
  - If the behaviour is unknown, it cannot be tested
- Under programming by contract, this concept is quite clear
  - You can only test what is defined in the contract

Extreme Programming

- ***Extreme Programming*** is the idea that you only write as much code as necessary to pass the current set of tests
  - Tests are written before the code is implemented
  - Writing tests and code can be interleaved
  - Anything outside of the testing is ignored and not implemented

Provability

RMIT UNIVERSITY

What is Correctness?

- What does is mean for code to “be correct”?
  - Compiles?
  - Executes?
  - Doesn’t crash?

What is Correctness?

- Under programming by contract
  - Correctness means the code compiles with the contract
  - Anything outside the contract is irrelevant
    - So the could could still do strange things or crash!
- More generally, properties can be defined
  - The code is correct if and only if it satisfies these properties
  - A property could be, the code does not crash

Static Reasoning

- Rather than:
  - Executing code to know it is correct, or
  - Hoping that the code is correct
- ***Static Reasoning*** proves that code obeys given properties
  - These properties prove the code's correctness

Hoare Logic*

- Hoare logic is a formal logic-based method for static reasoning
- Uses
  - First Order Logic
  - Deductive Reasoning

- * (This topic is not assessed in COSC1076)

Week 8 Online Quiz

RMIT UNIVERSITY

Week 8 Online Quiz

- On Canvas
  - 2 hour time limit
  - One attempt
  - That is, do this in one sitting
  - Marks are not available until everybody finishes the quiz
- Available from Monday of Week 8
  - You have until the following Sunday to complete the quiz
  - You need to find 2 hours in week 8 to do the quiz
- Questions will be based on
  - Self-revision questions
  - Tutorials & Labs

Assignment 2

RMIT UNIVERSITY

RMIT
UNIVERSITY