# Derivation of Algorithms

## Lecture 0
## Introduction

COMP H 4018
Lecturer: Stephen Sheridan



# Introduction

- The ability to prove mathematically that a program correctly implements its specification is increasingly important in ensuring that high integrity computer-based systems for security and safety-critical applications perform correctly.
- Program specification ≠ Program verification
- The formal verification of a program involves proving (in some way) the program's consistency with a formal specification. To do this, the meaning of the program also needs to be defined and understood. Thus it is necessary to provide a precise mathematical semantics for constructs of the programming language.



# Introduction

- These lectures are based on the work of Edsger Dijkstra, Tony Hoare, David Gries and many others who developed a formal model for constructing programs during the 1970's.
- Their belief, based on logic and mathematics, was that programs were amenable to proof by the laws of logic. Using what he termed weakest pre-condition semantics, Dijkstra proved that the statements of a programming language could be formally specified and that these formal definitions could be used to construct programs that are guaranteed to meet their specifications.



# Introduction

- For Dijkstra program testing was only useful to show the presence of bugs, but never to show their absence.
- Programming, in his view, is a scientific discipline and, as such, is governed by the mathematical laws of reasoning. In this course we shall study this mathematics.

# Software Disasters

- **Ariane 5 Flight 501**
  - **Flight 501**, which took place on June 4,1996, was the first test flight of the Ariane 5, expendable launch system. It was not successful. Due to a malfunction in the control software the rocket veered off its flight path 37 seconds after launch. It was torn apart by high aerodynamic forces which caused the onboard computer to trigger self-destruction.The break-up caused the loss of the payload: four Cluster mission spacecraft, resulting in a loss of more than $ 370 million.It is one of the most famous computer bugs in history.
  - http://en.wikipedia.org/wiki/Ariane_5_Flight_501



# Software Disasters

- **Mariner 1**
  - Mariner 1 was the first spacecraft of the Mariner program. Intended to fly by Venus, it failed during launch on July 22, 1962. A hardware failure in an antenna caused the booster to lose contact with guidance systems on the ground, so an onboard computer assumed control. However, that computer's software contained a bug.The error had occurred when an equation was being transcribed by hand in the specification for the guidance program. The writer missed the superscript bar in  (the nth smoothed value of the time derivative of a radius). Without the smoothing function indicated by the bar, the program treated normal minor variations of velocity as if they were serious, causing spurious corrections that sent the rocket off course. It was then destroyed by the Range Safety Officer.
  - http://en.wikipedia.org/wiki/Mariner_1



# Software Disasters

- **Therac-25**
  - Therac-25 was a radiation therapy machine produced by Atomic Energy of Canada Limited (AECL) and CGR of France after the Therac-6 and Therac-20 units. It was involved with at least six known accidents between 1985 and 1987, in which patients were given massive overdoses of radiation, which were in some cases on the order of hundreds of grays. At least five patients died of the overdoses. These accidents highlighted the dangers of software control of safety-critical systems, and they have become a standard case study in health informatics.
  - http://en.wikipedia.org/wiki/Therac-25



# Software Disasters

- **Virtual Case File**
  - Virtual Case File, or VCF, was a software application developed by the United States Federal Bureau of Investigation between 2000 and 2005. The project was not close to completion when it was officially abandoned in January 2005, having turned into a complete fiasco for the FBI. In addition to wasting at least US$100 million, the failure brought widespread criticism to the Bureau and its Director Robert S. Mueller III.
  - http://en.wikipedia.org/wiki/Virtual_Case_File

# Software Disasters

- **Y2K38**
  - The year 2038 problem may cause some computer software to fail before or in the year 2038. The problem affects programs that use the POSIX time representation, which represents system time as the number of seconds since January 1, 1970. This representation is standard in Unix-like operating systems and also affects software written for most other operating systems because of the broad deployment of C. On most 32-bit systems, the time_t data type used to store this second count is a signed 32-bit integer. The latest time that can be represented in this format, following the POSIX standard, is 03:14:07 UTC on Tuesday, January 19, 2038. Times beyond this moment will "wrap around" and be represented internally as a negative number, and cause programs to fail, since they will see these times not as being in 2038 but rather in 1901. Erroneous calculations and decisions may therefore result.
  - http://en.wikipedia.org/wiki/Year_2038_problem



# Tools and Techniques

- During this course we will use a combination of **Propositional Logic**, **Predicate Calculus** and a notional called the **Guarded Command Language**.
- Propositional calculus and predicate calculus along with the laws of logic and natural deduction will form the basic building blocks that will allow us to write program specifications and proofs in the guarded command language.
- The idea here is that we will derive a programs verification from its specification.



# Tools and Techniques

- We will employ a number of mathematical techniques to derive program proofs.
  - Taking the conjuncts as the invariants
  - Replacing constants with variables
  - Invariant diagrams
- We will discuss and practise each of the above techniques in more detail later in the course.
- Each technique listed above could not operate without the use of **ASSERTIONS**.



# Assertions

- “Summary: an assertion is a Boolean formula written in the text of a program, at a place where its evaluation will always be true or at least, that is the intention of the programmer. In the absence of jumps, it specifies the internal interface between the part of the program that comes before it and the part that comes after.The interface between a procedure declaration and its call is defined by assertions known as preconditions and post-conditions. If the assertions are strong enough, they express everything that the programmers on either side of the interface need to know about the program on the other side, even before the code is written. Indeed, such strong assertions can serve as the basis of a formal proof of the correctness of a complete program.”
  - Tony Hoare Senior Researcher, Microsoft Research Ltd., Cambridge, England. 2001.
  - **http://research.microsoft.com/~thoare/6Jun_assertions_personal_perspective.htm**

```java
import java.util.Scanner;

public class Assertion
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a number between 0 and 10: ");
        int number = input.nextInt();

        // assert that the absolute value is >= 0
        assert (number >= 0 && number <= 10) : "bad number: " + number;

        System.out.printf("You entered %d\n", number);
    }
}
```



# Guarded Command Language

- During this course we will derive complete programs that use the guarded command notation.
- This notation was developed by Edsger Wybe Dijkstra and is used extensively in the the recommended text that accompanies this course.
- **Programming: the derivation of algorithms**, Anne Kaldewaij, Prentice-Hall International Series in Computer Science, 1990.



# Guarded Command Language

- The guarded command language will be used to write program specifications and full solutions.
- The notation itself is very similar in syntax to C, C++ or Java and can be very easily translated into real code in any of the above.
- Therefore, any programs derived as part of this course can be implemented in Java and tested.



# Guarded Command Language

- **Problem**
  - Given an array f[0..N) of integers, N > 0, find the smallest value & the frequency of its occurance. Specify and derive a solution.
- **Specification**

```
|[ Con N: int; {N > 0}
   f : array[0..N) of int;

  var
  x,k: int;

  S

  {x = min j: 0 ≤ j < N: f.j ∧ k = #j : 0 ≤ j < N: f.j = x}
]|
```

# Guarded Command Language

**Complete Solution**

```
|[ Con N: int; {N > 0}
  f : array[0..N) of int;
  var
  x,k,n: int;
  x, n, k := f.0, 1, 1
  {P0 ∧ P1}
  do n < N →
      {P0 ∧ P1 ∧ n < N}
      if f.n = x →
              k := k + 1;
      [] f.n > x →
              skip;
      [] f.n < x →
              x := f.n; k := 1;
      fi
      n := n + 1;
      {P0 ∧ P1}
  od
  {P0 ∧ P1 ∧ n = N}
  {x = min j: 0 ≤ j < N: f.j ∧ k = #j : 0 ≤ j < N: f.j = x}
]|
```



```java
public class SmallValFreq
{
    public static void main(String[] args)
    {
        final int[] array = {1,3,5,9,2,3,4,9,9,1};
        final int N = array.length;

        int x,k,n;
        x = array[0]; k=1; n=1;

        while(n < N)
        {
            if (array[n] == x)
            { k = k + 1;
            }
            else if (array[n] > x)
            { ; // skip do nothing
            }
            else if (array[n] < x)
            { x = array[n];
              k = 1;
            }
            n = n + 1;
        }
        System.out.println("Smallest value is " + x + " and its frequency is " + k);
    }
}
```



# Problem difficulty …

- As this is your first exposure to using formal methods to derive computer programs, we will only concern ourselves with the derivation of solutions to simple well defined problems with small state spaces.
- It would be impossible given the time constraints imposed by a semesterised course to derive solutions to large scale software systems.
- Therefore, we will concentrate on interesting small scale problems that can be understood without any ambiguity. We will use these problems to guide us through each of the formal programming techniques introduced as part of this course.



# Problem difficulty …

- Programs that require loops
  - Calculate 2 to the power of N, $2^N$
- Array processing problems
  - Find the index of the largest element in an integer array
- Loops within loops
  - Sum the columns of an integer array
- Searching and sorting
  - Sort the elements of an integer array using selection sort
- Partition problems
  - Given a character array containing only A's, B's, and C's. Partition the array so that all C's proceed all A's, and all A' proceed all B's.

# Course structure

- Introduction
- Propositional calculus, predicate calculus, laws of logic, laws of natural deduction
- Guarded command language
- Making program specifications
- Axiomatic semantics of the guarded command language
- How to prove a loop is correct
- Formal programming techniques
  - Taking conjuncts as invariants
  - Replacing constants with variables
  - Using invariant diagrams



# Course structure

- **Exam 70%**
- **CA 30%**
  - CA will be comprised of weekly work and an assignment
- **NOTE:** Given the mathematical nature of this subject it is **ADVISED** that all students attend each and every single lecture, lab and tutorial session.
- **NOTE:** you will not reach the level where you can pass the written exam in this subject without carrying out all of the worksheet and assignment exercises given as part of your continuous assessment!!
- **NOTE:** this subject requires lots and lots of practice!!
- **NOTE:** you cannot cram for this subject!!