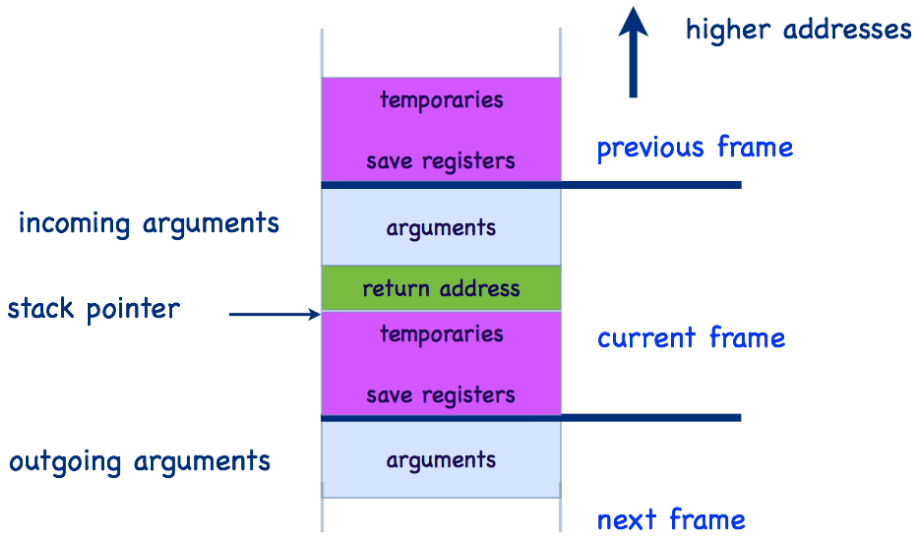
Swinburne University of Technology

School of Science, Computing and Emerging Technologies

LABORATORY COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Lab number and title:	8, Fundamental Stack Operations
Lecturer:	Dr. Markus Lumpe

Any sufficiently advanced technology is indistinguishable from magic.
Arthur C. Clarke



Lab 8: Fundamental Stack Operations

In this tutorial, we review the fundamental operations of a stack and define a `DynamicStack` class with automatic resizing.

A stack automatically adjusts its size to the number of elements it has to store. The size of a stack can change in two ways: it can increase or decrease. We use a standard heuristic for the changes: we double the size when we increase the stack size, and half the size when we decrease the stack size. This allows the amortized costs of all stack operations to remain O(1).

To control the size changes, we employ a load factor. Initially, an empty stack has a load factor of 0 (the stack contains one free slot). If a stack is full (its load factor is 1), we double its size. As a result, we obtain a stack with a load factor of ½. Half the stack is filled.

When removing elements from a stack, we may again reach a load factor of ½ (down from some higher value). However, shrinking the stack size in this instance is unwise. This could result in an undesired effect. The next operation can be a push, which requires an immediate increase in the size by doubling it. This process can repeat. All benefits of controlling expansion and contraction are lost. Hence, rather than contracting the space at ½, we let the load factor decrease to ¼. A halving of the space at load factor ¼ results in a new load factor of ½. This is the same value as for expansion. Half the stack is filled.

We do not yet define full copy and move semantics for data types. For this reason, we delete the copy operations. The compiler will not synthesize these features any longer. Objects of class `DynamicStack` cannot be copied, similar to stream objects.

Template Class Stack

We follow the example shown in class (plus a `size()` method):

```
template<typename T>
class DynamicStack
{
public:

  DynamicStack() noexcept;
  ~DynamicStack() noexcept;

  DynamicStack( const DynamicStack& aOther ) = delete;
  DynamicStack& operator=( const DynamicStack& aOther ) = delete;

  size_t size() const noexcept;

  std::optional<T> top() const noexcept;
  void push( const T& aValue ) noexcept;
  void pop() noexcept;

private:
  T* fElements;
  size_t fStackPointer;
  size_t fCurrentSize;

  void resize( size_t aNewSize );
  void ensure_capacity();
  void adjust_capacity();
};
```

You may add debug output to monitor resize events.

The test driver pushes 30 elements and pops them afterwards. It has to end with "Success".