Original Code for Stack Template
From: "Exception Handling: A False Sense of Security", by Tom Cargill

```
template <class T>
class Stack
{
  unsigned nelems;
  int top;
  T* v;

public:
  unsigned count();
  void push(T);
  T pop();

  Stack();
  ~Stack();
  Stack(const Stack&);
  Stack& operator=(const Stack&);
};

template <class T>
Stack<T>::Stack()
{
  top = -1;
  v = new T[nelems=10];
  if( v == 0 )
    throw "out of memory";
}

template <class T>
Stack<T>::Stack(const Stack<T>& s)
{
  v = new T[nelems = s.nelems];
  if( v == 0 )
    throw "out of memory";
  if( s.top > -1 ){
    for(top = 0; top <= s.top; top++)
      v[top] = s.v[top];
    top--;
  }
}

template <class T>
Stack<T>::~Stack()
{
  delete [ ] v;
}

template <class T>
void Stack<T>::push(T element)
{
  top++;
  if( top == nelems-1 ){
    T* new_buffer = new T[nelems+=10];
    if( new_buffer == 0 )
      throw "out of memory";
    for(int i = 0; i < top; i++)
      new_buffer[i] = v[i];
    delete [ ] v;
    v = new_buffer;
  }
  v[top] = element;
}
```

```
template <class T>
T Stack<T>::pop()
{
  if( top < 0 )
    throw "pop on empty stack";
  return v[top--];
}

template <class T>
unsigned Stack<T>::count()
{
  return top+1;
}

template <class T>
Stack<T>& Stack<T>::operator=(const Stack<T>& s)
{
  delete [ ] v;
  v = new T[nelems=s.nelems];
  if( v == 0 )
    throw "out of memory";
  if( s.top > -1 ){
    for(top = 0; top <= s.top; top++)
      v[top] = s.v[top];
    top--;
  }
  return *this;
}
```

Four Wrong Attempts at Binary Search

From: "Object Oriented Software Construction, 2nd Edition", by Bertrand Meyer

Code is in Eiffel. The @ operator is used for array access, := for assignment, = for testing equality, and // for integer division.

```
From
  i := 1; j := n
until i = j loop
  m := (i + j) // 2
  if t @ m <= x then
    i := m
  else
    j := m
  end
end
Result := (x = t @ i)
```

```
from
  i := 1; j := n; found := false
until i = j and not found loop
  m := (i + j) // 2
  if t @ m < x then
    i := m + 1
  elseif t @ m = x then
    found := true
  else
    j := m – 1
  end
end
Result := found
```

```
from
  i := 0; j := n
until i = j loop
  m := (i + j + 1) // 2
  if t @ m <= x then
    i := m + 1
  else
    j := m
  end
end
if i >= 1 and i <= n then
  Result := (x = t @ i)
else
  Result := false
end
```

```
from
  i := 0; j := n + 1
until i = j loop
  m := (i + j) // 2
  if t @ m <= x then
    i := m + 1
  else
    j := m
  end
end
if i >= 1 and i <= n then
  Result := (x = t @ i)
else
  Result := false
end
```