# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM
2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

MCS 360 Lecture 41
Introduction to Data Structures
Jan Verschelde, 4 December 2017

# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

# The Final Exam

- Monday 11 December, BSB 337, from 8AM to 10AM.
- The exam is closed book. No calculators, no computers.
- This review has the focus on the first 18 lectures, the material before the first midterm exam:
  1. Lectures 1 to 18, the first six chapters in the textbook.
  2. ADT, vectors and lists, stacks and queues.
  3. Focus on data structure concepts, not so much on C++ programming.
- Questions on this review are representative, but the list is by no means exhaustive.
- Review the quizzes, homework problems, and midterm exams.

# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

# write an ADT

1. Write an ADT specification of time, represented by hours, minutes, and seconds.
   Time is kept in a 24-hour format, e.g.: 20:00:00 is 8PM.
   Give a value definition of time.
   Define operations to see whether any two given times are equal and whether one time is later than another given time.
   Define the addition and subtraction of times.
   Subtraction of two times is only allowed if the first time is later than or equal to the second one.
   List all relevant conditions.

# ADT for time

```
abstract Time;
/* Time is a tuple of 3 nonnegative integer
   numbers: (hours, minutes, seconds);
   The hours are between 0 and 23.
   The minutes are between 0 and 59.
   The seconds are between 0 and 59. */
```

The equality of two times:

```
abstract bool is_equal ( Time t1, Time t2 );
postcondition: is_equal(t1, t2) is true if
  t1.hours == t2.hours, t1.minutes == t2.minutes,
  and t1.second == t2.seconds;
  otherwise, is_equal(t1, t2) is false.
```

# ADT for time continued

```
abstract bool is_later ( Time t1, Time t2 );
postcondition: is_later(t1, t2) is true if
  t1.hours > t2.hours, or
  t1.hours == t2.hours, and t1.minutes > t2.minutes, or
  t1.hours == t2.hours, t1.minutes == t2.minutes, and
  t1.seconds > t2.seconds;
  otherwise, is_later(t1, t2) is false.

abstract Time operator+ ( Time t1, Time t2 );
postcondition: after t3 = t1 + t2 we have
  t3.seconds = (t1.seconds + t2.seconds ) modulo 60,
  t3.minutes = (t1.seconds + t2.seconds )/60
             + (t1.minutes + t2.minutes ) modulo 60,
  t3.hours = (t1.minutes + t2.minutes )/60
           + (t1.hours + t2.hours ) modulo 24.
```

# ADT for time continued

To define the subtraction of two times,
we convert the times to seconds, subtract,
and convert the difference to hours, minutes, seconds.

```
abstract Time operator- ( Time t1, Time t2 );
precondition: is_later(t1, t2).
postcondition: after t3 = t1 - t2 we have
  t3.seconds = (a1 - a2) modulo 60,
  t3.minutes = ((a1 - a2 - t3.seconds)/60) modulo 60
  t3.hours = (a1 - a2 - t3.seconds - t3.minutes)/3600
  where
  a1 = (t1.hours*60 + t1.minutes)*60 + t1.seconds,
  a2 = (t2.hours*60 + t2.minutes)*60 + t2.seconds.
```

# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

# $O(\cdot)$ notation

2. Show that $O(\log_{10}(n))$ is $O(\log_2(n))$.

# $O(\cdot)$

2. Show that $O(\log_{10}(n))$ is $O(\log_2(n))$.
   - A function $f(n)$ is $O(log_2(n)$ if there is a constant $c$, independent of $n$ such that $f(n) \leq c \cdot log_2(n)$.
   - Let $m$ be the constant: $10 = 2^{\log_{10}(m)}$, then $\log_2(10) = \log_{10}(m)$, $m = 10^{\log_2(10)}$, and $\log_{10}(m) = \log_2(10)$.

     Consider:

     $$n = 10^k = (2^{\log_{10}(m)})^k = 2^{k \log_{10}(m)} \quad \Rightarrow \quad \log_2(n) = k \log_{10}(m)$$
     $$\Rightarrow \quad \log_2(n) = k \log_2(10).$$

   - Therefore, $\log_2(n) = \log_{10}(n) \log_2(10)$, or $\log_{10}(n) = \frac{1}{\log_2(10)} \log_2(n)$.

# analysis of code

3. Consider the code below:

```
double x[n+1];
double v,y;
for(int i=0; i<n+1; i++)
   for(y=1.0, int j=0; j<n+1; j++)
      if(j != i)
         y = y*(v - x[j])/(x[i] - x[j]);
```

   1. What are the preconditions on the numbers in the array `x`? Use sufficient `assert` statements to enforce the preconditions.
   2. Give the loop invariant for the inner loop controlled by `j`. Describe using the proper mathematical terminology what the value for `y` represents.
   3. Count the number of arithmetical operations executed by the code for any value of `n`.
   4. Use the big $O(\cdot)$ notation to bound the magnitude of the cost of the code in function of `n`.

# enforcing preconditions with `assert`

Observe the statement

```
y = y*(v - x[j])/(x[i] - x[j]);
```

may divide by zero if `x[i] - x[y] == 0.0`.

```
double x[n+1];
double v,y;
for(int i=0; i<n+1; i++)
   for(y=1.0, int j=0; j<n+1; j++)
      if(j != i)
      {
         assert(x[i] - x[j] != 0.0)
         y = y*(v - x[j])/(x[i] - x[j]);
      }
```

# looking at the inner loop

```
for(y=1.0, int j=0; j<n+1; j++)
   if(j != i)
      y = y*(v - x[j])/(x[i] - x[j]);
```

computes a product, in mathematical notation:

$$y = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{v - x[j]}{x[i] - x[j]}.$$

At step $k > 0$: $y = \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{v - x[j]}{x[i] - x[j]} * \frac{v - x[k]}{x[i] - x[k]}.$

# counting operations

```
for(int i=0; i<n+1; i++)
   for(y=1.0, int j=0; j<n+1; j++)
      if(j != i) y = y*(v - x[j])/(x[i] - x[j]);
```

Examining the loop counters:

- In the outer loop, `i` runs from 0 to `n`.
- In the inner loop, `j` runs from 0 to `n`, but excludes `i`.

The innermost instruction is executed $(n+1)n$ times.

The instruction `y = y*(v - x[j])/(x[i] - x[j]);`
involves 2 subtractions, 1 division, and 1 multiplication,
adding up to 4 arithmetical operations.
The arithmetical operation count is thus $4(n+1)n = O(n^2)$.

# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

# from single to double linked list

4. A node in a single linked list with item type `T` as template is

```
struct Node
{
   T data;  // T is template parameter
   Node *next; // pointer to next node

   Node(const T& item, Node* ptr = NULL) :
        data(item), next(ptr) {}
};
```

   1. Describe an algorithm to convert a single linked list to a double linked list.
   2. Write C++ code to define the conversion.

# from STL vector to STL list

5. Given an STL vector of doubles, write the code to construct an STL list with the same content as the given vector.

# looping through an STL list

6. Define a function that takes on input a nonempty STL list of integers and that returns the largest element of that list.

# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

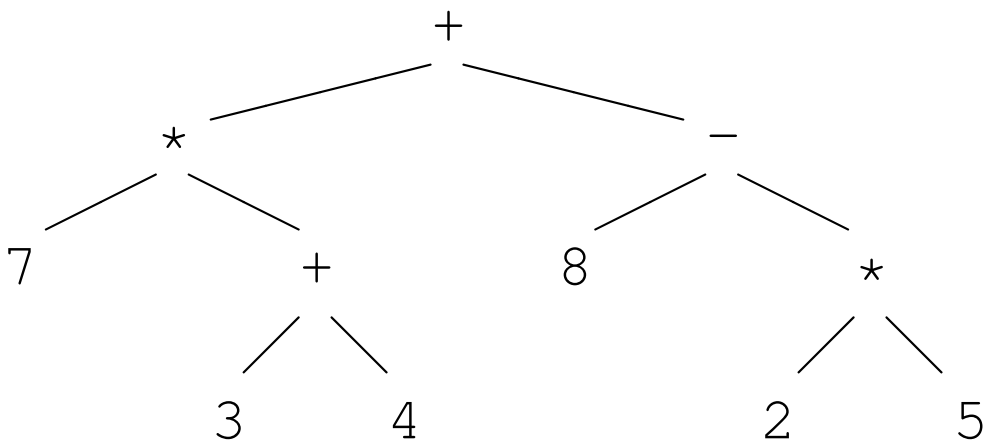
2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

# evaluation of a postfix expression

7. Consider the following postfix expression:
   `7 3 4 + * 8 2 5 * - +.`
   1. Draw the tree which represents this expression.
   2. Simulate the evaluation of the postfix expression using a stack. Draw all intermediate states of the stack.

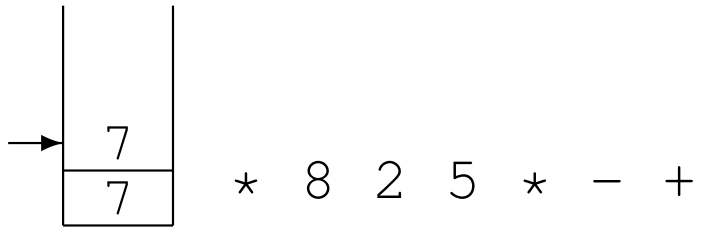
# constructing the tree of a postfix expression

7. Consider the following postfix expression:
   `7 3 4 + * 8 2 5 * - +.`
   1. Draw the tree which represents this expression.

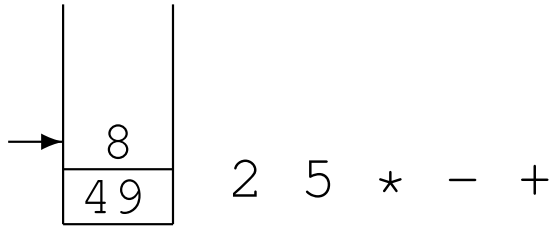


# evaluation of a postfix expression

7. Consider the following postfix expression:
   7 3 4 + * 8 2 5 * − +.
   1. Simulate the evaluation of the postfix expression using a stack. Draw all intermediate states of the stack.

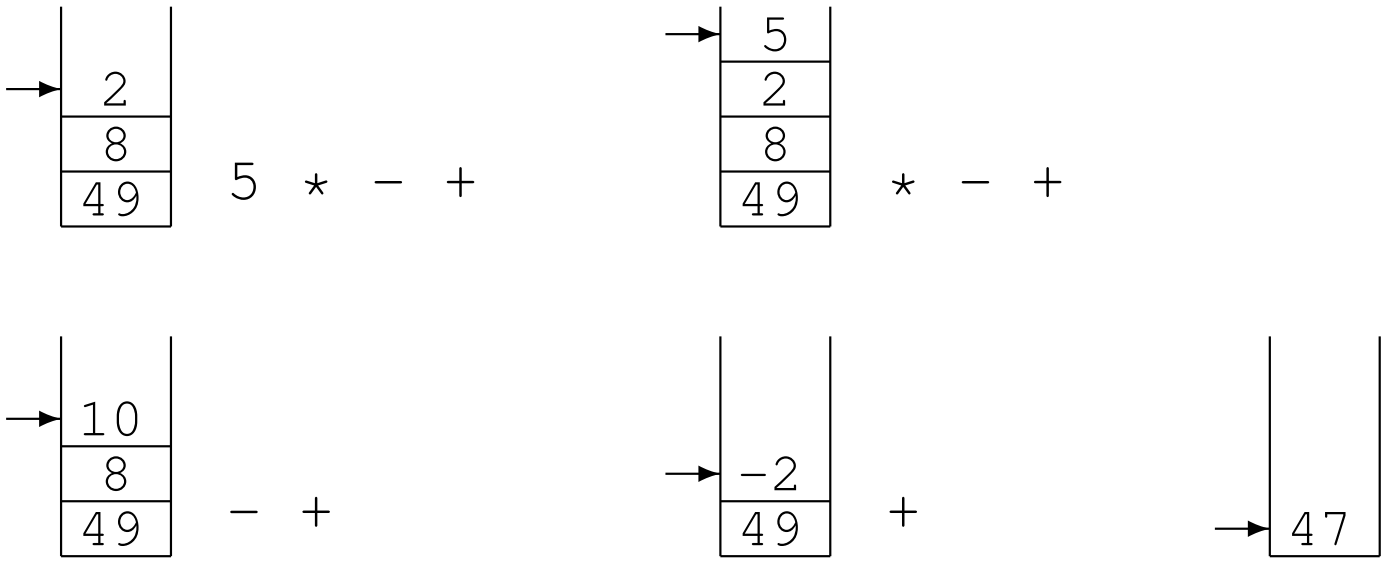







# evaluation of a postfix expression continued

7. Consider the following postfix expression:
   `7 3 4 + * 8 2 5 * - +.`
   1. Simulate the evaluation of the postfix expression using a stack. Draw all intermediate states of the stack.

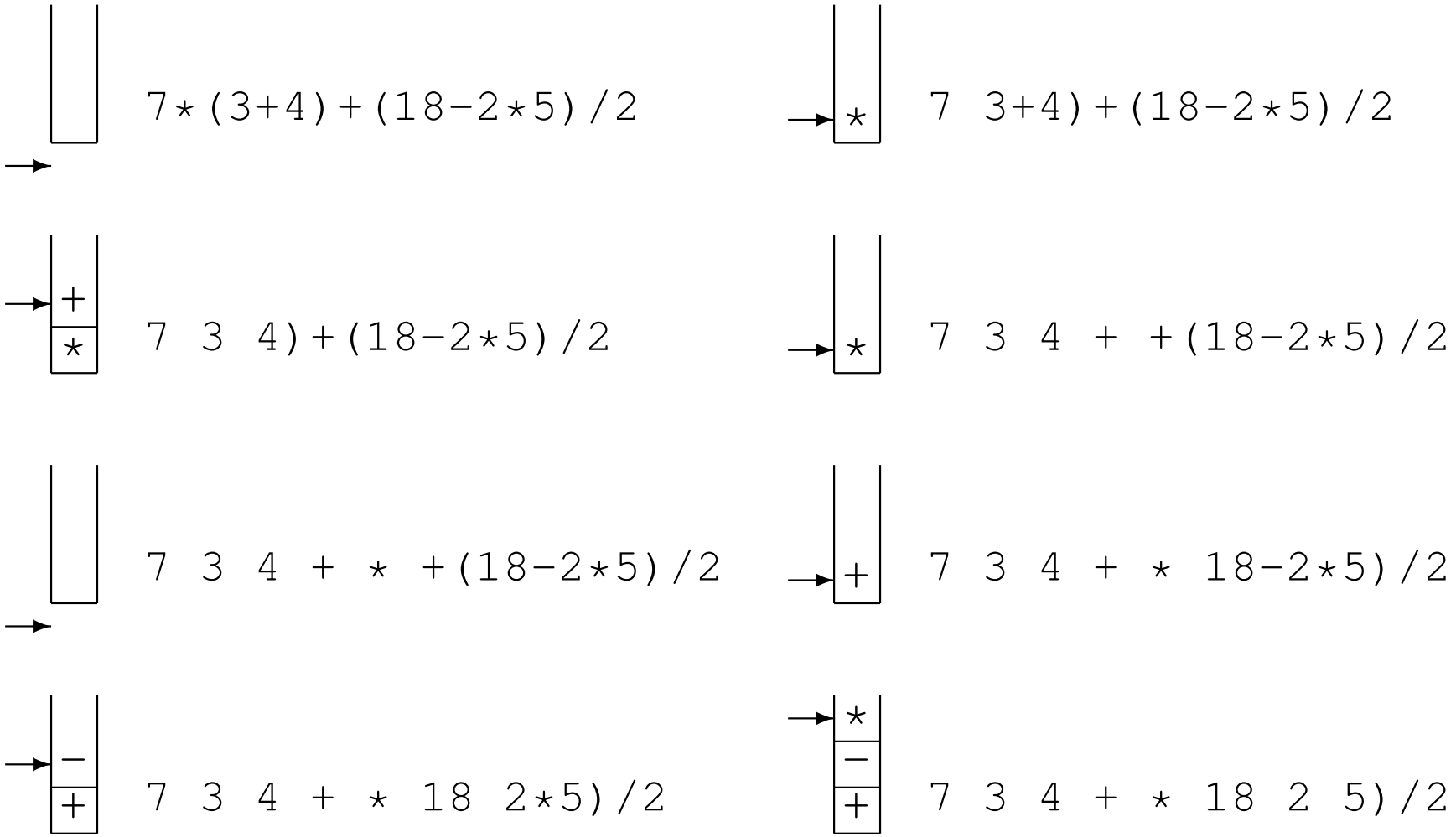


# converting to a postfix expression

8. Use a stack to convert `7*(3 + 4) + 8 - 2*5`
   to a postfix expression.
   Draw all intermediate states of the data structures used.

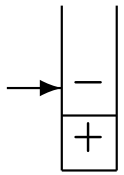
# converting to a postfix expression

8. Convert `7*(3 + 4) + (18 - 2*5)/2` to a postfix expression.

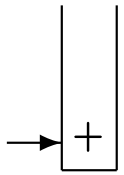


# converting to a postfix expression continued

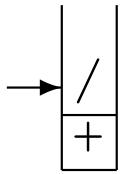
8. Convert `7*(3 + 4) + (18 - 2*5)/2` to a postfix expression.



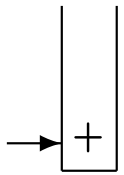

`7 3 4 + * 18 2 5 *)/2`




`7 3 4 + * 18 2 5 * - /2`



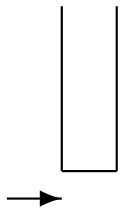

`7 3 4 + * 18 2 5 * - 2`




`7 3 4 + * 18 2 5 * - 2 /`

# converting to a postfix expression, the result

8. Convert `7*(3 + 4) + (18 - 2*5)/2` to a postfix expression.



`7 3 4 + * 18 2 5 * - 2 / +`

# stack as STL vector

9. Describe an implementation of a stack using an STL vector.
   Would you pop from the front or from the back?
   Justify your answer.

# Review of the First 18 Lectures

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - abstract data type
   - $O(\cdot)$ and code analysis
   - lists
   - stacks
   - queues

# priority queues

10. Assume we want to implement a priority queue with a single linked list. For every item in the queue we store a pair of doubles: the arrival time and the size of the job.
    1. Write a complete definition of the priority queue.
    2. Describe an algorithm to insert a job in any given queue. Items in the queue are sorted in increasing order on arrival time. Do not make assumptions on the arrival time of the item to insert. The priority queue may be empty.
       Illustrate your algorithm by drawing a general case inserting an item into a queue of at least five elements.
    3. Write C++ code to implement the algorithm described above.

# use STL for a priority queue of jobs

11. Describe a solution to the previous exercise using the STL. Which data structure of the STL would you choose? Justify your choice.