Lecture 2

Algorithms & Data Structures

Goldsmiths Computing

October 8, 2018

Outline

Outline

Lecture

- Module information
  - including ground rules
- Lab environment
  - 3 operating systems
  - 2 programming languages
  - as many text editors / IDEs as people
- Pseudocode
  - sequences
  - if

VLE activities

Programming language choice

- 93 have made their choice; thank you!
    - contact me directly if you need to change
- (about 50 have not: please make your choice now!)

VLE activities (cont'd)

Early Access

47 have elected to be able to access lecture materials early

- remember: early access correlates with worse performance

VLE activities (cont'd)

pseudocode quiz

Statistics so far:

- 315 attempts: average mark 6.18
- 110 students: average mark 6.90
  - 5 under 4.00
  - 15 at 10.00

Quiz closes at 16:00 on Friday 13th October

- no extensions
- grade is
  - 0 (for no attempt)
  - $30 + 70 \times (\text{score}/10)^2$

Reading

- CLRS, section 2.1
- DPV, sections 0.1, 0.2

Practical work

- installation and familiarization with lab environment
- downloading the lab bundle and running tests

Note:

- this week's labsheet is now available
- this week's lab session includes an assessment

Outline

For loops

To loop with a variable bound to a series of numbers, use **for** with a description of the series.

```
x ← 0
for 0 ≤ i < 100 do
    x ← x + 1
end for
```

what is the value of x after this?

For loops

To loop with a variable bound to a series of numbers, use **for** with a description of the series.

$x \leftarrow 0$
for $0 \leq i < 100$ **do**
  $x \leftarrow x + 1$
end for

what is the value of x after this? 100

For loops

To loop with a variable bound to a series of numbers, use **for** with a description of the series.

> $x \longleftarrow 0$
> **for** $0 \leq i < 100$ **do**
> $x \longleftarrow x + i$
> **end for**

what is the value of x after this?

For loops

To loop with a variable bound to a series of numbers, use **for** with a description of the series.

$x \leftarrow 0$
for $0 \le i < 100$ **do**
    $x \leftarrow x + i$
end for

what is the value of x after this? 4950

For loops

The order might matter: start with the left-hand bound and move towards the right-hand one.

$x \longleftarrow 0$
for $0 \leq i < 100$ **do**
    $x \longleftarrow i$
end for

$x \longleftarrow 0$
for $100 > i \geq 0$ **do**
    $x \longleftarrow i$
end for

what is the value of x after each of these?

For loops

The order might matter: start with the left-hand bound and move towards the right-hand one.

$x \longleftarrow 0$
for $0 \leq i < 100$ **do**
    $x \longleftarrow i$
end for
	$x \longleftarrow 0$
	for $100 > i \geq 0$ **do**
	    $x \longleftarrow i$
	end for

what is the value of x after each of these?

99	0

For loops

Use **continue** to proceed directly to the next iteration of the innermost loop, and **break** to finish the innermost loop

```
x ← 0
for 0 ≤ i < 10 do
    x ← x + 1
    if x > 3 then
        break
    end if
    x ← x + i
end for
```

```
x ← 0
for 0 ≤ i < 10 do
    x ← x + 1
    if x > 3 then
        continue
    end if
    x ← x + i
end for
```

what is the value of x after each of these?

For loops

Use **continue** to proceed directly to the next iteration of the innermost loop, and **break** to finish the innermost loop

```
x ← 0
for 0 ≤ i < 10 do
    x ← x + 1
    if x > 3 then
        break
    end if
    x ← x + i
end for
```

```
x ← 0
for 0 ≤ i < 10 do
    x ← x + 1
    if x > 3 then
        continue
    end if
    x ← x + i
end for
```

what is the value of x after each of these?

4

11

Nested loops

Loops nest: with nested loops, for each iteration of an outer loop, do the whole inner loop:

$x \longleftarrow 0$
for $0 \leq i < 3$ **do**
  for $0 \leq j < 4$ **do**
    $x \longleftarrow x + 1$
  end for
end for

what is the value of x after this?

Nested loops

Loops nest: with nested loops, for each iteration of an outer loop, do the whole inner loop:

$x \longleftarrow 0$
for $0 \leq i < 3$ **do**
  for $0 \leq j < 4$ **do**
    $x \longleftarrow x + 1$
  end for
end for

what is the value of x after this? 12

Forall

Iterate over members of a collection using **forall**

$x \longleftarrow 0$
for all $p \in$ prime numbers below 10 **do**
  $x \longleftarrow x + 1$
end for

what is the value of x after this?

Forall

Iterate over members of a collection using **forall**

$x \leftarrow 0$
for all $p \in$ prime numbers below 10 **do**
    $x \leftarrow x + 1$
end for

what is the value of x after this? 4

ordering

There may be a natural order to the iteration (*e.g.* when iterating over a linear collection), but usually there won't be. Don't rely on a particular order!

While

Use **while** to express a loop which tests a condition at the start of a sequence, and if that condition is true does another iteration of the loop.

$x \leftarrow 0$
$y \leftarrow 3$
while $y > 0$ **do**
  $x \leftarrow x + 1$
  $y \leftarrow y - 1$
end while

what value does x have after this?

While

Use **while** to express a loop which tests a condition at the start of a sequence, and if that condition is true does another iteration of the loop.

$x \leftarrow 0$
$y \leftarrow 3$
while $y > 0$ **do**
  $x \leftarrow x + 1$
  $y \leftarrow y - 1$
end while

what value does x have after this? 3

Repeat

Use **repeat until** to express a loop which tests a condition at the end of a sequence, and if that condition is false does another iteration of the loop.

x ← 0
y ← 3
repeat
  x ← x + 1
  y ← y − 1
until y < 0

what value does x have after this?

Repeat

Use **repeat until** to express a loop which tests a condition at the end of a sequence, and if that condition is false does another iteration of the loop.

> $x \leftarrow 0$
> $y \leftarrow 3$
> **repeat**
> $x \leftarrow x + 1$
> $y \leftarrow y - 1$
> **until** $y < 0$

what value does x have after this? 4

Loop

Use **loop** to express an unconditional loop. (You will need to use **break** to terminate the loop).

```
x ← 11342
loop
    if x = 1 then
        break
    else if x is even then
        x ← x ÷ 2
    else
        x ← 3 × x + 1
    end if
end loop
```

Function calls

Functions have zero or more arguments, and return one result. Call them using their name, with arguments in brackets

$n \leftarrow 5$
$x \leftarrow \textsc{fact}(n)$

Functions

Define functions using **function**, and return a value using **return**.

```
function FACT(n)
    if n = 0 then
        return 1
    else
        return n × FACT(n−1)
    end if
end function
```

Pre- and post-conditions

Make it clear to the reader what conditions a function requires to operate correctly, and what it does if those conditions are met

Require: $n \in \mathbb{N}_0$
Ensure: Compute and return n!

```
function FACT(n)
    if n = 0 then
        return 1
    else
        return n × FACT(n−1)
    end if
end function
```

Outline

Motivation

- useful abstraction of memory
- basic building block

Definition

A vector is a finite fixed-size sequential collection of data.

finite a vector has a non-negative integer length;

Definition

A vector is a finite fixed-size sequential collection of data.

finite a vector has a non-negative integer length;

fixed-size the length of the vector is immutable;

Definition

A vector is a finite fixed-size sequential collection of data.

finite a vector has a non-negative integer length;

fixed-size the length of the vector is immutable;

sequential a vector has a defined and static storage order of its elements;

Definition

A vector is a finite fixed-size sequential collection of data.

finite a vector has a non-negative integer length;

fixed-size the length of the vector is immutable;

sequential a vector has a defined and static storage order of its elements;

collection a vector's contents represents data in the collection; things not stored in the vector are not in the collection.

Operations

length	return the number of elements in the vector
select[k]	return the k^{th} element of the vector
store![o,k]	set the k^{th} element of the vector to o

Operations

length	return the number of elements in the vector
select[k]	return the k^{th} element of the vector
store![o,k]	set the k^{th} element of the vector to o

In pseudocode, respectively:

length	LENGTH(v)
select[k]	v[k]
store![o,k]	$v[k] \leftarrow o$

Operations

length	return the number of elements in the vector
select[k]	return the k^{th} element of the vector
store![o,k]	set the k^{th} element of the vector to o

In pseudocode, respectively:

length	LENGTH(v)
select[k]	v[k]
store![o,k]	$v[k] \leftarrow o$

Constructor:

- **new** Vector(n)

Operations

length	return the number of elements in the vector
select[k]	return the k^{th} element of the vector
store![o,k]	set the k^{th} element of the vector to o

In pseudocode, respectively:

length	LENGTH(v)
select[k]	v[k]
store![o,k]	v[k] $\leftarrow$ o

Constructor:

- **new** Vector(n)

length

Not Vector operations

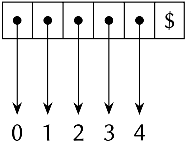
delete!

insert!

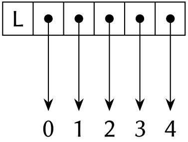
resize!

Implementation

sentinel (C strings)



length-data (everything else)



Complexity analysis

How much work do operations take?

select, store!

1. offset calculation: base address + k × element size
2. pointer read (select) or write (store!)

Complexity analysis

How much work do operations take?

select, store!

1. offset calculation: base address + k × element size
2. pointer read (select) or write (store!)

length

Depends on implementation strategy:

sentinel	1. initialize count to 0, position to 0 2. iterate position through string, incrementing count, until the sentinel ($) 3. return count
length-data	1. read the length slot

Work

1. reading
    - CLRS 10.3
    - Poul-Henning Kamp, “The most expensive one-byte mistake”, ACM Queue 9:7, 2011

Outline

Motivation

- can (in principle) build all record types out of pairs
- basic building block

Definition

A pair is a 2-tuple of data

tuple an ordered collection

Definition

A pair is a 2-tuple of data

tuple: an ordered collection

2-: with exactly two elements

Operations

left	return the left element of the pair
right	return the right element of the pair
set-left![o]	set the left element of the pair to o
set-right![o]	set the right element of the pair to o

Operations

left	return the left element of the pair
right	return the right element of the pair
set-left![o]	set the left element of the pair to o
set-right![o]	set the right element of the pair to o

In pseudocode, respectively:

left	LEFT(p)
right	RIGHT(p)
set-left![o]	LEFT(p) $\longleftarrow$ o
set-right![o]	RIGHT(p) $\longleftarrow$ o

Operations

left	return the left element of the pair
right	return the right element of the pair
set-left![o]	set the left element of the pair to o
set-right![o]	set the right element of the pair to o

In pseudocode, respectively:

left	LEFT(p)
right	RIGHT(p)
set-left![o]	LEFT(p) $\longleftarrow$ o
set-right![o]	RIGHT(p) $\longleftarrow$ o

Constructor:

- **new** Pair(l, r)

Operations

left	return the left element of the pair
right	return the right element of the pair
set-left![o]	set the left element of the pair to o
set-right![o]	set the right element of the pair to o

In pseudocode, respectively:

left	LEFT(p)
right	RIGHT(p)
set-left![o]	LEFT(p) $\longleftarrow$ o
set-right![o]	RIGHT(p) $\longleftarrow$ o

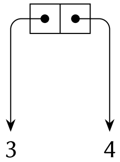
Constructor:

- **new** Pair(l, r)

right

left

Implementation



Complexity analysis

left, set-left!, right, set-right!

1. pointer read (left, right) or write (set-left!, set-right!)

Complexity analysis

left, set-left!, right, set-right!

1. pointer read (left, right) or write (set-left!, set-right!)

constructor

1. fixed-size (two-word) allocation
2. two pointer writes

Higher-cardinality tuples

(a,b,c)	((a,b),c)
(a,b,c,d)	(((a,b),c),d)
(a,b,...,z)	...((a,b),...z)

Work

1. Implement a pair data structure in Java or C++.
2. Using your pair data structure, implement a triple data structure, with operations first, second, third, corresponding setters, and a constructor with three arguments.
3. Consider the implementation of an N-tuple from pairs. What is the time and space overhead, in terms of N, for the implementation presented in the lecture?
4. Can you come up with an implementation of N-tuples from pairs with a lower time cost? What is the space overhead cost of this implementation?

Activities over the next week

1. Finish pseudocode quiz (16:00 Friday 12th October)
2. Selection of programming language (16:00 Friday 12th October)
3. Start pairs and vectors quiz
4. Labsheet 02, including assessment
   - deadline: by 16:00 Friday 12th October
5. Reading
6. Tuple implementation strategies
7. Answer and ask questions on the forum