

# Intro to Data Structures & Algorithms

Wagner Truppel  
Lecturer, Dept. of Computer Science & Engineering  
UC Riverside

[wagner@cs.ucr.edu](mailto:wagner@cs.ucr.edu)  
<http://www.cs.ucr.edu/~wagner>

<http://www.cs.ucr.edu/cs14>

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 1

---

---

---

---

---

---

---

---

# Announcements I

- Email us using **only** your UCR address
  - ◆ If you're not a CS or ENG major, use the account created for you in the lab
  - ◆ Example: `cs14ab@cs.ucr.edu`
  - ◆ TAs and I will **not** reply to messages with non-UCR addresses
- Email with missing information
  - ◆ Please make your messages easy for us to deal with so that you can get a prompt reply
  - ◆ Include your **complete name**, **student ID**, the **course you're taking**, your **lecture section**, and your **lab section**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 2

---

---

---

---

---

---

---

---

# Announcements II

- We're **not** using ilearn/Blackboard
- Class mailing list
  - ◆ See the course web page for directions
  - ◆ You **must** add yourself to the list
  - ◆ Read it frequently!
  - ◆ You can only sign in with a UCR address
  - ◆ Use the list to ask questions and interact with us and with other CS 14 students
  - ◆ **No** postings containing code

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 3

---

---

---

---

---

---

---

---

## Today's Topics

- Testing ADTs
- C++ Exceptions
- Algorithm Quality
- Introduction to Complexity Theory
- Review of C++ Templates

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 4

---

---

---

---

---

---

---

---

## Testing ADTs

- Suppose I give you some code and claim that it performs some function. How can you verify my claim ?
- Example: I give you a **Counter** class and I claim it does indeed implement the **ADT Counter** interface. How can you prove that it does ?
  - ◆ You *could* look at the source code...
- What if the source code isn't available ?
  - ◆ You can verify adherence to the ADT *interface* but that isn't enough
  - ◆ Verifying adherence to the *full* ADT is more difficult

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 5

---

---

---

---

---

---

---

---

## Testing ADTs

- To verify *full* compliance to an ADT you need to **test** the **axioms** defined by the ADT
- Example: **ADT Counter axioms**
  - ◆ `get(new()) = 0`
  - ◆ `get(inc(c)) = get(c) + 1`
- How do you test the axioms ?
 

```
Counter c = Counter();
if (c.get() == 0)
    print "Fine.";
else
    print "Test failed.";
int old = c.get();
c.inc();
int new = c.get();
if (old + 1 == new)
    print "Fine.";
else
    print "Test failed.";
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 6

---

---

---

---

---

---

---

---

## Testing ADTs

- Still not enough, however...
  - ◆ Testing can *only* show the **presence** of errors, **not** their **absence** !
- The more you test, the more confident you become
- Two kinds of test environments
  - ◆ Black-box test
    - \* Use only the ADT axioms to come up with test cases
  - ◆ White-box test
    - \* Look at the source code to come up with test cases
- In this week's lab you'll have a chance to implement and test the ADT Counter

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 7

---

---

---

---

---

---

---

---

## C++ Exceptions

- Say that you're writing a program that reads and writes files
- When the program is running, it could encounter the following problems
  - ◆ can't read a file
  - ◆ can't write to a file b/c it's locked
  - ◆ can't even *find* the darn file
- Since you know these problems could happen, you safeguard your program by adding **error handling**

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 8

---

---

---

---

---

---

---

---

## C++ Exceptions

- Error handling in the past
  - ◆ set an error code and check it elsewhere
  - ◆ return an error code and check it at the caller level
  - ◆ just ignore the error and pray that things will work
- Is there a better way?
  - ◆ If you get a problem (expected or not) and you know how to fix it, then fix it right there and then !
  - ◆ If you get a problem (expected or not) and you **don't know** how to fix it, pass it to the caller
  - ◆ Ok... but how do you *pass an error* ?
  - ◆ Easy... if errors are represented as data types, you **throw** them

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 9

---

---

---

---

---

---

---

---

## C++ Exceptions

```
double f(double a, double b) {
    if (b == 0.0d)
        throw std::domain_error("Div by zero!");
    else
        return (a / b);
}
```

- You should always throw exceptions when you expect potential problems whose fault isn't your own
- You should throw exceptions **only** for exceptional cases
- Exceptions in C++ and in Java belong to a class hierarchy
- For now, use `std::domain_error`

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 10

---

---

---

---

---

---

---

---

## C++ Exceptions

```
try {
    ...
    // code that might raise an exception
    // such as opening a file that doesn't exist, etc
}
catch (std::exception &e) {
    // here you can fix the problem if you know how
    // and have enough information to do so
    // you can also get information on the exception
    std::cout << e.what() << std::endl;
}
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 11

---

---

---

---

---

---

---

---

## Algorithm Quality

- Effectiveness
  - ◆ we can actually use them !
- Correctness
  - ◆ hey, if it's broken, why use it ?
- Finiteness
  - ◆ it'd better terminate !
- Efficiency
  - ◆ memory and time are precious
- Understandability
  - ◆ if no one can understand your algorithm, what's the point in writing it ?
- Maintenance
  - ◆ someone's gotta do it...

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 12

---

---

---

---

---

---

---

---

## Complexity Theory

- Deals with **efficiency only**
- **Runtime** efficiency: simple instructions are assumed to take **constant time**
  - ◆ Additions & subtractions
  - ◆ Assignments
  - ◆ Simple comparisons
- We're not saying that they all take the exact same amount of time – they surely don't !

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 13

---

---

---

---

---

---

---

---

## Complexity Theory

- We're not *even* saying that they each take the same amount of time in different computers – they don't !
- All we're saying is that the time each of those operations takes does **not** grow with the size of the input
 

```
for (int i = 0; i < n; i++)
    { a[i] = 0; }
```
- We say that the above loop has a runtime complexity of  $O(n)$  ["Order n"]
- "Order" means we don't care about specific numbers but only about general behavior

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 14

---

---

---

---

---

---

---

---

## Complexity Theory

int i = 0;	O(1) steps	constant time
binary search	$O(\log_2 n)$ steps	logarithmic growth
array init	$O(n)$ steps	linear growth

$O(1)$  is faster than  $O(\log_2 n)$  which is faster than  $O(n)$

*Always faster?* No... for small values of n, it may be the case that the order is violated. But we never really care about small n values.

Three complexity cases:  
**worst case, best case, average case**

Runtime complexity depends on the **input** to the algorithm and **not just** on the algorithm itself

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 15

---

---

---

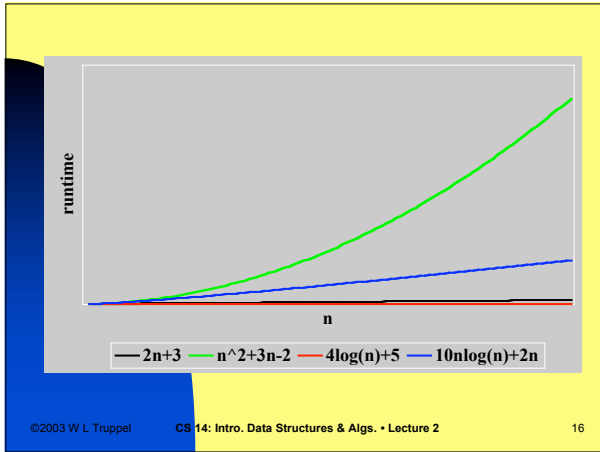
---

---

---

---

---




---

---

---

---

---

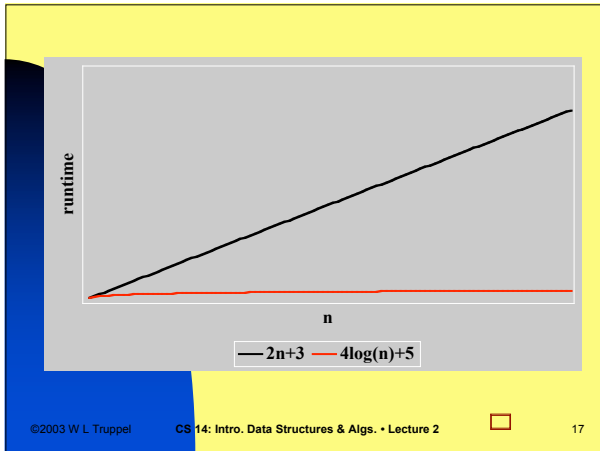
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

## Complexity Theory

- Example: linear search
  - ◆ **Best** case: you get lucky and find it the very first time you try...  $O(1)$  !
  - ◆ **Worst** case: you get **un**lucky and have to check **every** case...  $O(n)$
  - ◆ **Average** case: you only have to check about half the number of elements...  $O(n/2) = O(n)$   
(b/c constants don't matter...)
- Runtime complexity depends on the **input** to the algorithm and **not just** on the algorithm itself

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 18

---

---

---

---

---

---

---

---

---

---

## Complexity Theory

- **Best** case: the algorithm executes only the **minimum** number of steps needed to complete its task
- **Worst** case: the algorithm executes the **maximum** number of steps needed to complete its task
- **Average** case: the complexity obtained by averaging over the complexities achieved for random inputs to the algorithm

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 19

---

---

---

---

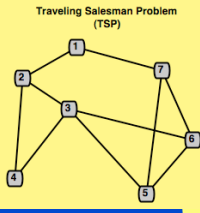
---

---

---

---

## Complexity Theory



Traveling Salesman Problem (TSP)

- What we'd *always* like to have:  $O(1)$
- Yeah, well, dream on... what we often get are (these are fine)
  - ◆  $O(n)$ ,  $O(n \log_2 n)$
- Sometimes, we also get (great !)
- ◆  $O(\log_2 n)$
- But other times, we have to bite the bullet and use algorithms that are
  - ◆  $O(n^2)$ ,  $O(n^3)$ , or even worse
- There are problems for which the only *exact* algorithms known have an exponential runtime complexity
  - ◆ They're the worst of the lot

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 20

---

---

---

---

---

---

---

---

## C++ Templates

- Suppose you want to find the maximum of two **integer** values
- You could write a function for that:
 

```
int max(int a, int b)
{ if (a > b) return a;
  else return b; }
```
- Suppose now that you want to find the maximum of two **float** values
- You could write another function for that:
 

```
float max(float a, float b)
{ if (a > b) return a;
  else return b; }
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 21

---

---

---

---

---


---

---

---

## C++ Templates

- Suppose next that you want to find the maximum of two **double** values
- You could write a function for that too:  
`double max(double a, double b)`  
`{ if (a > b) return a;`  
 `else return b; }`
- What if you want to find the maximum of two **Speed** values?
- You could also write a function for that:  
`Speed max(Speed & a, Speed & b)`  
`{ if (a > b) return a;`  
 `else return b; }`



©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 22

---

---

---

---


---

---


---

---

## A simple puzzle...



- Say you have a glass of beer and another glass, of red wine.
- Now suppose that for some reason, you want to swap the contents of the two glasses.
- How do you do it?



©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 23

---

---

---

---

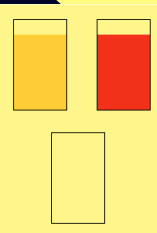
---

---

---

---

## A simple puzzle...



- Say you have a glass of beer and another glass, of red wine.
- Now suppose that for some reason, you want to swap the contents of the two glasses.
- How do you do it?
- You need **another** glass...
- Does it matter that the glasses had beer and wine?

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 24

---

---

---

---

---

---

---

---

## Another puzzle...

- How do you swap two *integers* ?
- The integers are stored in two variables
- But you need a **third** one...

```

void swap(int & var1, int & var2)
{
    int temp = var1;
    var1 = var2;
    var2 = temp;
}
    
```

- Does it matter that you have two integers as opposed to two values of another type?

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 25

---

---

---

---

---

---

---

---

## A general swap function

- We'd like to be able to say:
  - ◆ Give me two values of some type **T** and I'll swap them with this function

```

void swap(T & var1, T & var2)
{
    T temp = var1;
    var1 = var2;
    var2 = temp;
}
    
```

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 26

---

---

---

---

---

---

---


---

## No problem, right ?

- We could simply write this:
 

```

void swap(T & var1, T & var2)
{
    T temp = var1;
    var1 = var2;
    var2 = temp;
}
            
```
- It doesn't work !
  - ◆ because T would be interpreted as the name of some class we defined, rather than as a **parameter**



©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 27

---

---

---

---

---

---

---

---

**Function templates**

## No problem, right ?

- We need to tell the compiler that **T** is a **parameter** and not an actual type. How?

```
template<class T>
void swap(T & var1, T & var2)
{
    T temp = var1;
    var1 = var2;
    var2 = temp;
}
```

- Despite the keyword **class**, you can now use this function for **any type that supports the assignment operator**.
- `swap(int1, int2); // swap int variables`
- `swap(char1, char2); // swap char variables`
- `swap(foo1, foo2); // swap Foo variables`

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 28

---

---

---

---

---

---

---

---

## Class templates

- We've seen **function templates** (also referred to as template functions)
- We can also have **class templates** (never referred to as template classes)
- Same basic idea, same basic syntax

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 29

---

---

---

---

---

---

---

---

## Class templates

```
template<class T>
class Pair
{
private:
    T first;
    T second;
public:
    Pair(T frst, T sec);
    T getFirst() const;
}

template<class T>
Pair<T>::Pair(T frst, T sec)
{ first = frst; second = sec; }
```

- Inside the class, just use **T** as if it was some specific type (class or not)
- Member functions are then template functions

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 30

---

---

---

---

---

---

---

---

## Class templates

```
template<class T>
class Pair
{
    private:
        T first;
        T second;
    public:
        Pair(T first, T sec);
        T getFirst() const;
}
```

- How to use it? Just as before, but now you need to specify what type to use:
- `Pair<int> point(2, 7);` // a two dimensional point
- `Pair<Lens> glasses(Lens(), Lens());` // a pair of lenses

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 31

---

---

---

---

---

---

---

---

## Cranky Compilers

- Many compilers don't like templates very much
- Make sure to put your template definitions in the same file where they're used, and **before** they're used

©2003 W L Truppel CS 14: Intro. Data Structures & Algs. • Lecture 2 32

---

---

---

---

---

---

---

---