31251 – Data Structures and Algorithms

Week 1, Autumn 2020

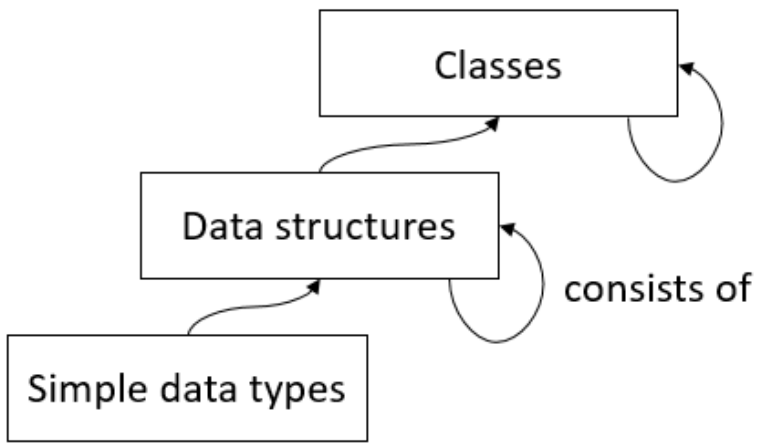
Xianzhi Wang

What are Data Structures and Algorithms?

What's Data Structure?

A way to store and organise (non-persistent) data.

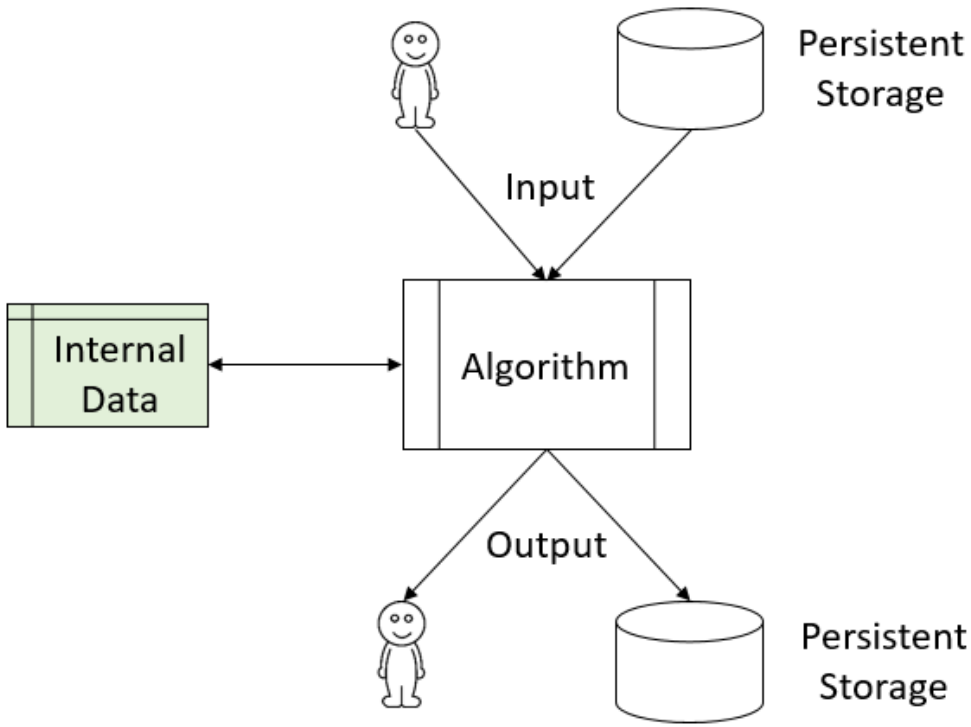
- Simple Data Types: int, string, float, etc.
- Data Structures: a set of data items with definite associations
- Classes: data structure + operations



What's Algorithm?

- A well-defined computational process that takes some input and produces an output.
- A tool for solving a well-specified computational problem.
- You don't actually need a computer for them though.

- Problem := Input + (Expected) Output
- Program := Data Structure + Algorithm
- Evaluation
  - Algorithm: validity/correctness, accuracy, efficiency
  - Data Structure: usually assessed as a part of algorithm



But you've seen this before!

- All the little patterns you learnt in Applications Programming (48024) are algorithms or templates for algorithms.
- All the code you've written was made up of algorithms (at least the bits that worked).
- What we're really doing in this subject is starting to build a formal awareness of algorithms as separate from computer programs.

Some Adminstrative Matters

- UTSOnline
  - Subject Outline
  - Announcements
  - My Grades
  - Staff Contacts
- Ed
  - Lessons: slides, practices
  - Discussion: online Q&A
  - Sway: quizzes release & submission
  - Assessments: assignments release and submission
  - Challenges: interesting to try
  - Online C++ compiler

Lecture Recording

- Both lectures might be delivered live via Zoom.
- I will make supplementary videos as necessary.

Assessment

- 3 short quizzes
  - 10 multiple-choice questions each
  - Weeks 3, 6 & 9: Monday (released), Friday (due)
  - Weight: 10%
- 2 programming assignments
  - Assignment 1: Week 6 (released), Week 8 (due)
  - Assignment 2: Week 9 (released), Week 11 (due)
  - Weight: 25% + 35%
- 1 open-book exam
  - 20 multiple-choice questions
  - 4 short-answer questions
  - Weight: 30%

U:PASS

- Get extra help from a student who did well in the subject.
- Places are limited—the people attend, the more slots they offer.
- Sign up: http://www.tinyurl.com/upass2020

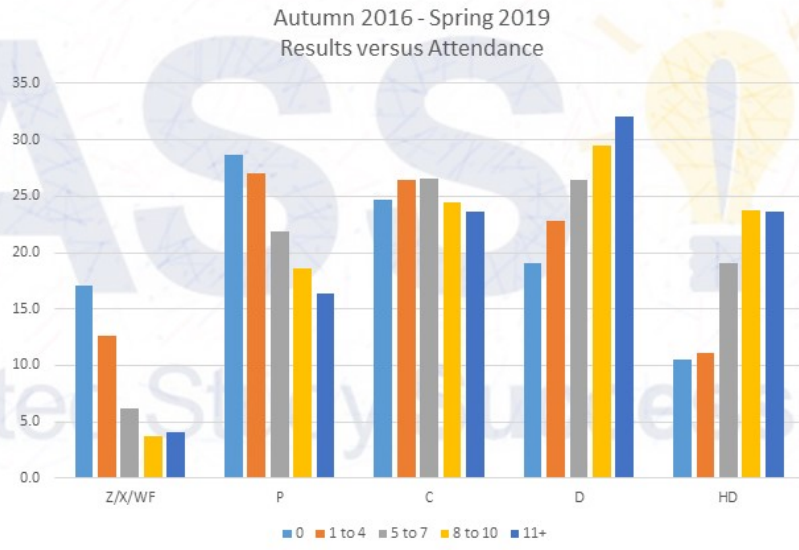
U:PASS
UTS Peer Assisted Study Success

It is a great environment for asking questions without getting the feeling of being judged (U:PASS student, 2019)

- Do better (on average)
- Get help from other students
- Make friends
- Actually study

Session times:
www.tinyurl.com/upass2020

U:PASS film:
www.tinyurl.com/upassfilm



U:PASS
UTS Peer Assisted Study Success

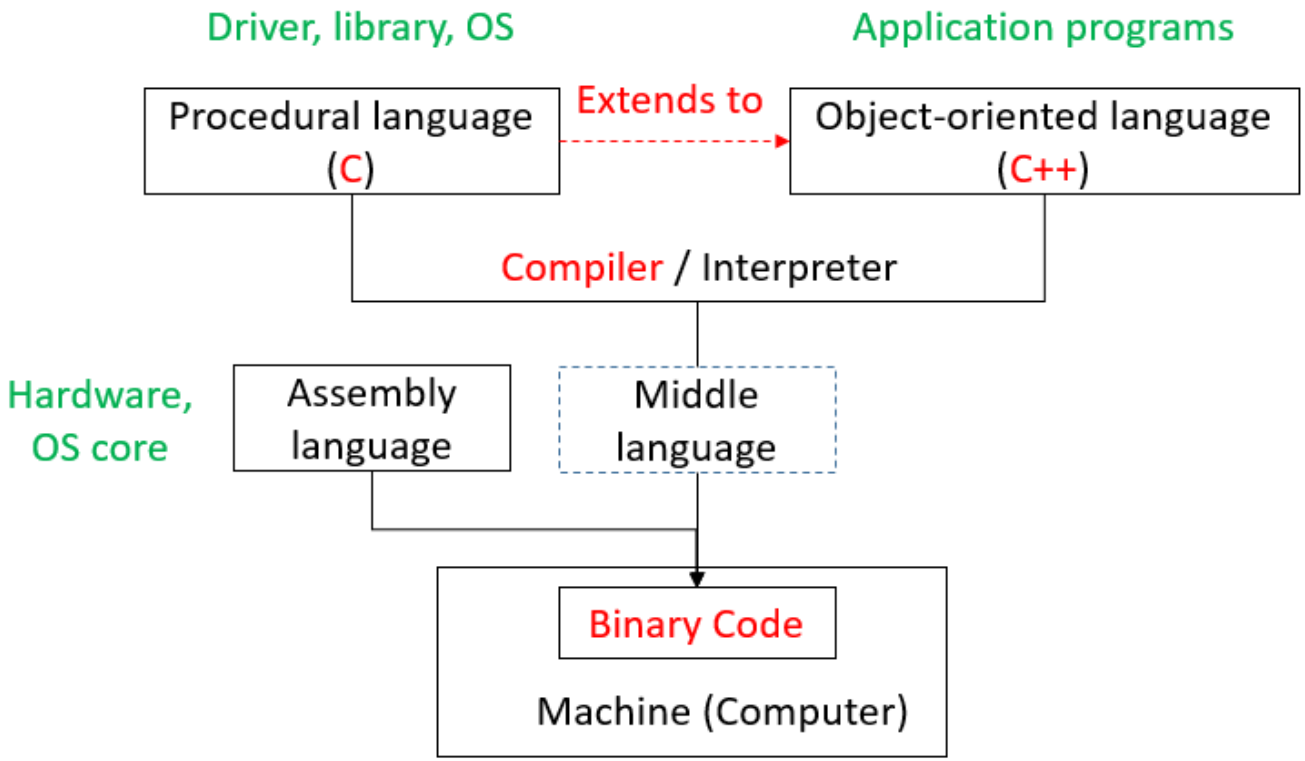
Consultation

- Coordinator
  - 2:00pm-4:00pm Thursday, starting from Week 2
  - FEIT learning precinct (Building 11, Level 5, Room 300)
- Tutors
  - See timetable for Cmp1.01 through Cmp.17
  - Time and place vary for different activity numbers
- Online
  - Ed's Discussion Board
  - I will regularly check questions each week

Now Some C++

Why C++ in this Subject

- Broad application, highly efficient, scalable
- Better manipulation of memory space and data structures



And now for something complete different.

Today's topics:

- Strings (briefly)
- Arrays, or "who thought this was a good idea?"
- Pointers
  - References vs Pointers
  - Dereferencing a pointer
- Classes
- Headers and Source files, or "at least this isn't as bad as arrays"
- Lists and Linked Lists! (Yay, an actual data structure)

- Strings in C are just null terminated `char` arrays.
  - Aside: ... what is null in C++?
  - Mostly just `0`, or something that looks like it (yay C).
  - Since C++11, an actual `null_ptr` type exists, so you can have a proper null that isn't just `0`.
- Where could that possible go wrong?
  - What if you forget the null?
  - What if you want to know the length?

C++ has a proper `string` class (`std::string`) that conceptually wraps a `char[]` and fixes these problems:
`http://www.cplusplus.com/reference/string/string/`

Arrays in C++

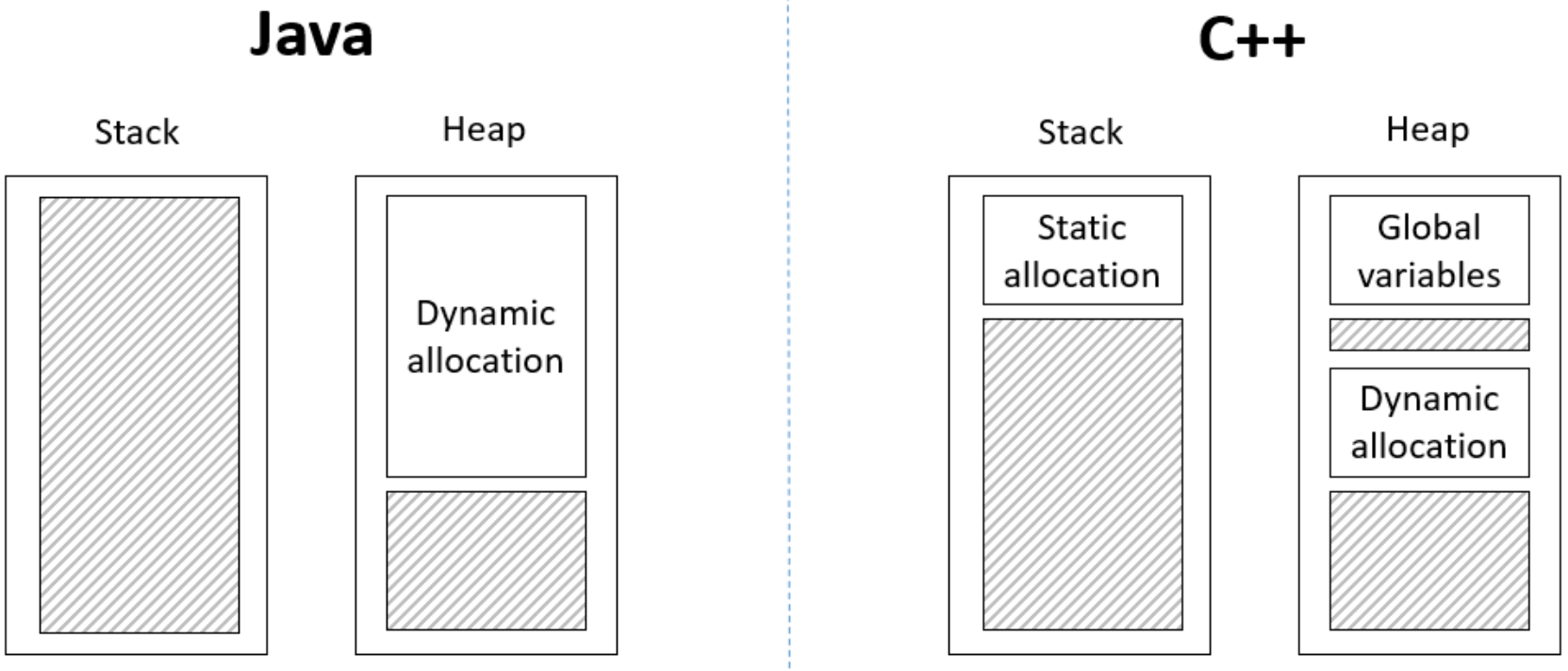
- Arrays in C++ look a lot like Java arrays:
  - `int a[4] = {1,2,3,4};`
  - `int a[] = {1,2,3,4};`
  - `int a[4] = {};`
  - `int a[4];`
- Note that these are all statically created.
- ..huh? What does that mean?

Static and Dynamic Memory Allocation

- C++ has a more complex allocation system than Java (at least from the programmer's perspective).
- Things can be statically allocated:
  - They are automatically deallocated when they go out of scope.
  - What does this mean for return data?
- Or dynamically allocated:
  - Created on the heap with the `new` keyword.
  - C++ has no garbage collection, so you have to free space by yourself.
- Short version, don't use `new` unless you mean it!

C++ vs. C vs. Java

- C: call inherent functions, e.g., `malloc()`, `free()`
- C++: call constuctors/deconstructors, e.g., `new`, `delete`



- In all the previous array examples, the size was known at declaration.
- The program does its own memory management – you need to know the size!
- What if we don't know the size?
- Arrays decay to pointers to the first element.
  - So an `int[]` can be treated as a `int*`.
  - Wait, what's a pointer?

Pointers!

- Pointers are what make C++ programming annoying!
- Actually they're not so bad, they're just variables that tell you where something is in memory.
- Pointers point to something.

Pointers!

- To create a pointer to type `t`:
  - `t *foo;`
  - The spaces around the * don't matter (*i.e* `t* foo`, `t * foo` and `t *foo` are all the same).
- A pointer is really just a number that is the (start) address of a simple/complex object in computer memory.
- To get what it's pointing at, we *dereference it*:
  - `t bar = *foo;`
  - If you're derefencing to get a member `(*foo).m`, you can write the alternative `foo->m`, as C++ tries to reduce pointer use!
- To get the address of something, use the address operator:
  - ```
    int foo = 5;
    int *bar = &foo;
    ```
    ```

References!

- C++ also has references.
- References are like pointers, but:
  - They can't change where they point to once initialised.
  - They're transparently dereferenced:
- They're created with the & operator:
  - `int & foo = ...;`
- But then they work like the thing at the other end:
  - `foo = foo + 5` does what you'd expect.
  - What would it do to a pointer?
- References are good for passing data without copying it
  - This should be familiar from Java – it does essentially the same thing).

Back to Arrays

- So if we want to create an object of which we don't know the size (*e.g.*, as a parameter or return type), we need a pointer:
  - `int * tabulate(Data dataObject)...`
- But how do we know that we're getting what, e.g., an array?
- We don't!
- Well... that's not so good... how do we fix it?
- `std::vector`!
  - `vector<int> v;` where v is a variable of type Vector, which stores integer elements.

Classes in C++ look a lot like Java classes, where all classes inherit a public parentClass without declaration:

```
#include <string>

using std::string; // using namespace std;

class myClass {
  private:
    int privateInt;
  public:
    int getPrivateInt();
    void setPrivateInt(int newValue);
    string toString();
};
```

- Notice that the methods have no content there.
- They can, but they don't have to.
- C++ routinely separates definition from source code.
  - It expects a single pass compiler, so you have to have all the names in the right order!
- Typically definitions are put in *header* files (usually with a `.h` extension, but not necessarily).
- Source code is normally in source files (usually `.cpp`, but again, that can change).
- Sometimes code is put in the header file (sometimes it even makes sense to do so!).

So what do we do with header files if they have no code?

- Declare things in the right order for `#includes`.
- Create the equivalent of interfaces (virtual classes!)

A Data Structure!

A Data Structure

- We now have almost enough to build our first data structure!
- But first: Abstract Data Types
  - ADTs are specifications of behaviour of Data Types.
  - They don't specify *implementations*.
  - Adhereing to an ADT allows us to code without having to know implementation details (good for teams, reusability and modularity).
  - In Java, we'd achieve this with an `Interface` and abstraction.

The `List` ADT

- A list stores data in a sequential order.
- So what methods should a list have?
  - Something to check if it's empty?
  - Something to add to the front of the list?
  - Something to add to the end?
  - Something to get the first element?
  - Something to get the rest of the list?
- We should be able to manage that!

An intList Abstract Class

```
class intList {
public:
  virtual intList(){}; // constructor
  virtual ~intList() {}; // destructor

  /* Below are virtual functions */
  virtual bool isEmpty() = 0;
  virtual void prepend(int c) = 0;
  virtual void append(int c) = 0;
  virtual int head() = 0;
  virtual intList tail() = 0;
};
```

Now the implementation (of some of it)

See C++ files.