SCC 110

LO1: Algorithms

Naive approach = inefficient with linear complexity

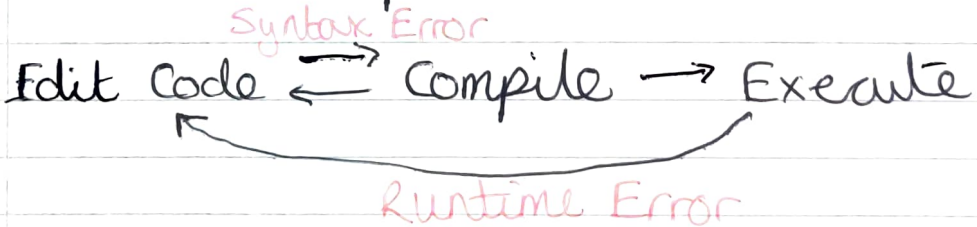
A parallel approach is good as it has logarithmic complexity

Sequence - Going down lines
Selection - If/Else statements
Iteration - Controlled loops

LO2:

All the key ideas are on scratch however there's no file i:o, or complex data structures and it's very slow to execute and inte

C is a high level language with a text editor and compiler



In C variables are typed and given different memory

LO3:

Loops repeat until condition is false
HOUSE RULES
If, else scanf("%d", &answer)
Decomposition

LO4: Reading compute programs

Comments, indentation, white space, variable names

Arrays - store multiple (related) instances of the same data type.
int myArray [3]
Arrays can be 2D myArray [2][2] = 2nd row 2nd column

LO5:

A function may return arithmetic, a structure, a union, a pointer, a void but not a function or array.

functions can be called upon. e.g z = timesbytwo(y)
Void means no arguments

Code:

ls - files pwd - working directory cd - change directory

Array search / sort for smallest:

```
variables int i, smallest = myArray[0] smallestPos = 0 largestPos
                                                    = 0
for (i = 0; i < 10; i++) {
    if (smallest > myArray[i]) {        // if not smallest
        smallest = myArray[i];          // make smallest
        smallestPos = i;
    }
}
```

15/11/2021

SCC 110 LO5

Multiple functions can be made and declared

direct-declarator (parameter-type-list)

A function may return an arithmetic type, a structure, a union, a pointer or void. But not a function or an array.

Array values aren't local to functions (special case)

15/11/2021

SCC 110 LO6

When you declare a variable you partition space in memory of a specific type.

Indirection -

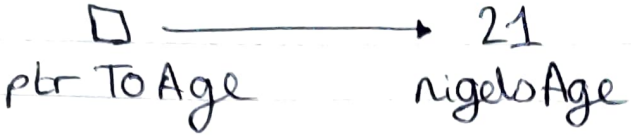
* Fundamental concept in all fields of computing

(The ability to reference something using a name, reference or container instead of the value itself)

Pointers -

Imagine we declare a variable x that contains an integer. How do we talk about the space labelled x (rather than contents of x)

Pointers give a level of indirection. We can adjust which variable they 'point to'



Pointers are just special variable types that hold the location of another variable

```
int x = 65;
int y = 32;
int *p = &x;
```

Declare a pointer to an int

Take a reference to x

e.g

```
int main (){
  int x;          Declares space for int
  int *p;         Declares space for pointer to int
  x = 7;          Puts 7 in x
  p = &x;         Puts address of x into p
  *p = 8          Puts 8 into p
```

%p = pointer value when printing

Slide 25

```
void changeScoreByOne (int *pScore) {
    *pScore = *pScore + 1; }

int main {
    int x = 0;
    changescore by one (&x);
    printf ("x = %d \n", x); }
```

This explains why & is needed before variables in scanf

```
void swap(int x, int y) {
  int temp;
  temp = x;
  x = y;
  y = temp;
}
```

swap(a, b); Swaps copies of a and b

```
void swap (int* px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```

swap(&a, &b); Swaps values of a and b ore

When to use a Pointer ?

* We can modify parameters passed into functions
* Very efficient to pass a pointer to data into functions
* We can use pointers to "iterate through" data.
* We can create our own efficient data structures.

SCC 110 LO7 - Strings

A string is really a sequence or an array of characters.

```
char *z = "hello";
```

h	e	l	l	o	\0
z[0]	z[1]	z[2]	z[3]	z[4]	z[5]

A string always needs to be given enough space in memory (in bytes)

```
char s[6] = "hello";
```

s = address / pointer to first character.

Strings are not immutable. Like an array we can modify it's elements.

e.g. `s[0] = 'y';`

Homemade strlen() function:
1) start at beginning of the string
2) check if character == '\0'
3) If not more onto next character
4) Loop from 2)

However we can use the library <string.h> for our string manipulation

<string.h> -

- int strlen(char*) - length of string s
- char *strchr(char*, int) - find a character within a string
- strcat(char*d, char*s) - append s to d.
- strcpy(char*d, char*s) - copy s to d.
- strcmp(a, b) a, b can be strings, variables or "text".
  If the same = 0
  If different = 1 or -1

SCC 110 L08 - Testing

Programs need to be tested systematically

Types of testing
- Testing by developers
- Testing by "testers"

Developers try and make sure the software works according to their understanding of the spec
Testers try and make sure the software works according to everyone's understanding of the spec.

White (glass) box testing mainly by developers
- How the program works and efficiency

Black box testing mainly by testers
- Testing for inputs and outputs

Simple Tests -
- Does it work for a simple representative set of inputs?
- Does it work for all possible inputs? (Boundaries)
- Does the output match the input?

Exercise: Identify test cases for string-length()
1) Works for strings of different length?
2) Returns 0 for 0 length strings?
3) Effect of non-ASCII characters?
4) Handling of a null-pointer

Black Box Cases

What makes a good test?

- Tests are designed to help find errors.
- Isolate and test certain expected behaviours.
- Repeatable and necessary.

SCC 110 L09 Structs + Dynamic Memory

Coursework Assessment 1 released to be complete before Week 10.

Variables can only represent simple, scalar values and are of a fixed size. They also have to worry about scope.

SCC 120 Compound Variables (structs in C) -
A struct is a user-defined grouping of variables

```
struct person {
  char name[20];
  int age;
  char gender;
};
```

Once declared we can create instances of structs
struct person client;
struct person janedoe;

You can also give them an [alias] to hide that they are structs.

typedef struct person TPerson;
TPerson lecturer = {"Adrian", 21, "m"}

Parts of structs are accessed like :
client.age, client.gender

When accessing a struct using a pointer use
-> because . has higher precedence than *

```
struct person *client;
client->gender = 'm';
```

You can also create an array of structs.

```
struct person clients[10];
client[0].age = 6;
```

Dynamic Memory
- Allows us to allocate memory when we need it, deciding the size as the program runs.
- Can use with pointers to create true dynamic and complex data structures.

malloc() allocates blocks of memory
free() releases memory for reuse.

```
char *str
str = (char*) malloc(sizeof(char) * 100);
```

(char*) — cast to a pointer of the right type

sizeof(char) — supply the number of bytes for a given type.

```
free(str);
```

pointer str still contains address of malloc'd block but this should not be reused.

```
str = NULL;
```

defensive programming.

SCC 110 L10
Debugging and Common Mistakes

Debugging -

* Read line by line
* Read as the computer executes
* Follow loops and branches
* Track variables and values

Use printf as the program runs to track variable values throughout the program.

→ Common Mistakes -

1) The NULL statement `for(i=0; i<10; i++);`
2) Loops that don't ever run (first case)
3) == and = are different.
4) scanf arguments and pointers. Arguments are pass by value and need a pointer
5) Accessing invalid array arguments
6) Assuming variables have been declared
7) Not enough storage.

7/1/2022

SCC 110 L11 (Java)

- Pros and cons of C

Java is an object-oriented programming language
- Originally designed for embedded devices.
- Language of choice for software integration.
- Platform Independent.
- Abstracts over hardware.

Java Virtual Machine executes it's own machine code known as bytecode.

Object-Oriented creates improved modularity and ease of code reuse. It also allows for development parallelism. Allow for scalability of large projects.

Language - (Based on C syntax) (similar to C++)

```
public class ClassName {
}
```
Classes are helpful for organisation

{ starts a class

```
public static void main(String [] arguments) {
    System.out.println("Hello World");
}
```
function main

array (another class)

System.out: object

println: print method

Java has strings and doesn't have true pointers.
One class per file in Java.
Main is always void (no return value)

```
1 javac FileName.java
2 java FileName
```
To run a java file.

14/11/2022

SCC 110 L12

Modularity relies on standards. The interface between a module and the world must be well defined

C is a procedural programming language. There is a well defined start (main function). Code is broken down to manageable chunks (functions). Procedural programming languages are structured around the code. They treat code and data as separate concepts.

Object Oriented programming languages combine code and data. This enables encapsulation and standard interfaces.

Object Oriented programming -
Objects provide encapsulation so data and code cannot be separated. Data is defined through attributes
Behaviour is defined through methods

Objects protect their inner workings with an API and interactions with objects occur through methods. The programmer only exposes what they choose to. This allows them to provide sealed, reusable boxes of code.

In Java objects are defined by classes. They use instance variables and methods to define an object. Instance variables are used to hold the state of an object and their values are persistent.

Methods are named, parameterized blocks of code (like functions) but are contained within a class.

return_type method_name (parameter_list)

Instantiating a class:

```
public static void main( String[] arguments ){
    Car c = new Car();
}
```

- Car: class name
- c: variable name
- new: builds the instance
- Car(): constructor for the new object *

Invoking methods on instances

<instance>.<method>(<parameters>)

e.g. c.drive(14)

Java has a set of pre-written classes

31/1/2022

SCC 110 L13 Encapsulation

Modularity relies on encapsulation. The module must contain it's inner workings, protect them and only expose functionality through its interfaces.

Access Control (Private and Public) -
When instance variables and methods are declared public they can be accessed from inside and outside the class. They can present to an API.
When declared private they can't be accessed from methods outside the class

As a point of good practice never create public instance variables in Java.

Accessors :
These are public methods that wrap access to a private instance variable. e.g

```
public class Pen {
    private String colour;
    public String getColour() {
        return colour;
    }
}
```

A mutator is the same but changes the value of a private instance variable.

A constructor helps initialise objects (no return type)
e.g

```
public class Pen {
    public Pen() {
        // Initialisation
    }
}
```

Constructes can have any number of parameters. These are then required when an instance is created with new.

```
public class Pen {
    public Pen (String col, int size) {
        colour = col;
        nibSize = size;
        inkLevel = 100;
    }
}

public class Driver {
    public static void main (String args) {
        Pen biro = new Pen ("Blue", 1);
        Pen marker = new Pen ("Black", 8);
    }
}
```

How to use classes by others.
Analyse available classes, Make/instantiate classes, use methods.

API Documentation.
Programmers document their classes. This is done using the JavaDoc standard. It is embedded using comment blocks. It documents public classes, methods and constructes

@param <name> documents meaning of parameter 'name'. @return documents meaning of a value returned from a method.

`javadoc -d doc *.java` creates a javadoc.

11/2022

SCC 110 L14 Git Version Control

Version control : local :

A locally stored repository : a database / backup of all the version history. A sandbox is a folder of the most recent version from the repository.

Version control : client - server

A repository is held centrally with locally held sandboxes. e.g CVS (Concurrent Version Systems), SVN (Subversion). Disadvantage : Point of failure.

Version control : client - server workflow

Checkout version of code from server into the sandbox. Modify files in the sandbox. Merge - update changes others may have made on the server. Add files to be committed. Commit changes back to the server with a message.

Version control : distributed

A repository is held centrally with local sandboxes and repositories. e.g Git, Mercurial. Advantage : Each client is a backup of the repository.

Version control : distributed workflow

Clone the repository from server or another client, modify files in the sandbox, add files to be committed, commit to local repository, fetch changes from each other, merge changes and push merged changes to the central repository.

Installing Git (PRACTICAL)

Linux: sudo apt-get install git

Remember password and username:

git config credential.helper 'cache --timeout=300'

Git commands (PRACTICAL)

git status git clone git commit -m "commit message"
git pull git fetch git branch
git merge git submodule git add git log
git push git blame git checkout
git fork (plagiarism?) git config

Practices:

Merge other people's work frequently
Commit as often as possible with useful messages

Managing development issues:

Issues are enhancements / features or bugs.
Github allows issues to be created, assigned and closed. This allows only one person at a time to work on an issue which reduces conflicts.

Branches -

A branch is a copy of the branch it came from.
Parallel development of the same code.
Git has a main branch.

Branches workflow:

Create an issue from main, make changes on the new branch. Create a pull request to merge changes into main branch. Merge changes.

Nobody can conflict with your changes until you merge them in.

Branches : Developer workflow

Clone a remote repository
Allocate an issue to themselves on issue tracker
Create a branch for the issue
Checkout the issue branch
Make changes to fix the issue
Commit changes
Create a pull request
Manage pull request by merging branch into the main branch.

SCC 110 L15 Type Composition

Variable Types
int, double, String for example.
Classes are types as well:
GameArena a = new GameArena(500, 500);

anywhere we can use a standard type we can also use a class.
e.g → Person[] people = new Person[10];
people[0] = new Person("Joe", 5);

This means classes can contain other classes

Composition:
When a class holds an instance variable that is of a class-type we call this composition.
(Another form of encapsulation)
Instances of classes can be stored in arrays, passed in method parameters, returned from a method and be instance variables.

Composition provides the ability to create increasingly more powerful classes, whilst maintaining modularity and keeping a lid on complexity.

21/2/2022

SCC 110 L16

Quiz from 9:00 Thursday.

GUIs in Java -
The swing package is very large and extremely flexible including textboxes, sliders, buttons and images.
It is heavily object-oriented.

GUIs are a standard set of components (windows, Buttons, text input areas, menus etc.)

GUI Components in Java -
Each type of component is a different class.
Implementation is therefore encapsulated.

Packages -
Swing classes reside in a package called javax.swing
A package is a collection of classes grouped together and they enable unique naming of classes.
Add `import javax.swing.*;` as the first line

JFrame -
JFrame represents a window in the host OS including the title bar, icons to minimise etc.
Two common forms: JFrame(), JFrame(String title)
(private) instance variables: boolean getVisible(); , void setVisible(boolean b); , String getTitle(); , void setTitle(String s); , void setSize(int x, int y);

JFrame instantiation -

Instantiate the JFrame class to create a new window. Windows by default are created in an invisible state.

```
JFrame a = new JFrame();   // Creates blank window
a.setVisible(true);        // Makes it visible
a.setTitle("Hello World"); // Window title
a.setSize(300, 300);       // Window size
```

JFrame closing

```
a.setDefaultCloseOperation(int operation);
```

int operations :

```
JFrame.EXIT_ON_CLOSE      // Terminates application
JFrame.DISPOSE_ON_CLOSE   // Closes window keeps app running
JFrame.DO_NOTHING_ON_CLOSE // Ignore
```

Point / Explanation,

JFrame.EXIT_ON_CLOSE uses an integer variable EXIT_ON_CLOSE that may look like this :

```
public static final int EXIT_ON_CLOSE = 1;
```

A static variable / method is called on a class not an instance. The variable is shared between all instances of that class.

A final variable cannot have it's value changed after being initialised and is used to define constants.

JFrames are responsible for managing the window not its content. To do that we use the JPanel class and it's components.

JPanel -

```
import javax.swing.*;
public class HelloWorld {
    JFrame a = new JFrame();
    JPanel panel = new JPanel();
    a.setContentPane(panel);
```

Components : JButton , JLabel , JTextField etc.

JLabel -

A read only field used for prompts, status messages etc

```
JLabel(String s);    // Plain Text
JLabel(Icon i);      // Image
```

Accessor and Mutators : getText() setText(String s).

JButton -

Clickable component for performing actions.

```
Constructors : JButton(String s); // Clickable Text
               JButton(ImageIcon i); // Clickable Image
```

JTextField -

User editable block of code used for collecting input

```
Constructors : JTextField(); // Created Empty
               JTextField(); // Created with Text
                             ↖ String s
```

Accessor and Mutators : getText() setText(String s)
setEditable(boolean) can enable/disable input.

Example -

```
import javax.swing.* ;
public class HelloWorld {
        JFrame a = new JFrame();
        JPanel panel = new JPanel();
        JButton b = new JButton("Press!");
        public HelloWorld {
            panel.add(b);
            a.setContentPane(panel);
            a.setTitle("Hello World!");
            a.setSize(300, 300);
            a.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
            a.setVisible(true);
        }
}
```

17/3/2022

SCC 110 L17 Swing GUI Layout + Asynchronous Program.

Non-trivial user interfaces -

Most GUIs require many components to be useful, Java specifically tries to prevent you from specifying where your components go. Layout managers do this to allow you to build GUIs that work on any computer / windowing system. They are a part of : `import java.awt.*;`

FlowLayout -

This is the simplest layout manager.

* Arrange components best to fit size of the container.
* Purely in left → right, top → bottom order
* Useful for simple GUIs

```
FlowLayout f = new FlowLayout();
panel.setLayout(f);        // Assigned to JPanel
```

Advantages and Disadvantages -

Highly Reactive (Dynamic reaction to panel size changes)
Easy to use however, lack of control over component placement. By default it centres components.

```
FlowLayout f = new FlowLayout(FlowLayout.LEFT);
FlowLayout f = new FlowLayout(FlowLayout.CENTER);
FlowLayout f = new FlowLayout(FlowLayout.RIGHT);
```

] for alignment

GridLayout -

Fixed size components in a matrix. Fits components into an n×m grid structure. Still left → right, top → bottom but on grid boundaries and can specify grid shape.

```
GridLayout layout = new GridLayout(int rows, int columns);
panel.setLayout(layout);
```

Advantages and Disadvantages -

Excellent for repeating components e.g mixing desk, numerical keypad. However can be too clinical for most applications. Still have not much control over which component goes where.

BorderLayout -

Relative positioning of upto 5 components.
Divides the container into N, S, E, W and Center where Compass points take priority over space.
Specify which area to put component inside the add method.
One component per location.

```
BorderLayout layout = new BorderLayout();
panel.setLayout(layout);
panel.add("North", okButton);
```

Setting your own Layout -

You can specify positions manually with a null Layout manager and setLocation and setSize methods. All swing components respond to these methods.

```
panel.setLayout(null);
okButton.setLocation(10, 10);
okButton.setSize(100, 25);
```

Multiple containers -

Panels can be placed inside other panels.

Sequential Programming -

So far all Java programs have been ...
This isn't good for modularity + efficiency.

main(String args[])
↓
Java code
↓
exit

Event-based programming -

A new programming paradigm, with no simple start to end flow of execution. Application registers interest in certain events (e.g button presses). Environment (Java) informs applications when they occur.

main(String args[])
↓
Java code
event ⇄ Java code] Event Handler
event ⇄ Java code]

Listener Interfaces -

Swing lets you register "Listeners" on GUI components. You register an interest in events you want to receive, provide an object with a well-known method which is invoked when that event occurs. These interfaces notify you when:

ActionListener: buttons are clicked
ChangeListener: sliders are moved
KeyListener: when a key is pressed (keyboard)
MouseListener: when a mouse button is pressed / moved
WindowListener: windows are resized / closed / minimised

Implement ActionListener -

Import java.awt.event package, implement ActionListener etc

```
import java.awt.event.*;
public class HelloWorld implements ActionListener {
    public HelloWorld() {
        someButton.addActionListener(this);
    public void actionPerformed(ActionEvent e) {
        // code to execute when button clicked
}
```

One ActionListener can register with multiple buttons :
ActionEvent can differentiate the source by responding to getSource() and returning the object that generated the event.

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == okButton)
        //...
    if(e.getSource() == cancelButton)
        //...
}
```

. .
. .

7/3/2022

SCC 110 L18 Debugging

Compile often, Test frequently, Add one feature at a time. Compiler messages tell you the file and line of code where the syntax error is found. Exception traces at runtime also do this. Exceptions often cascade.

If you are unable to find the bug by code inspection start logging diagnostic data about the program in each entry/exit using printf() / System.out.println()

Runtime Debuggers -

Allow you to see each line of code being ran in real time and see variable values change as the program is ran. INSTALL JAVA LANGUAGE EXTENSIONS.
Breakpoints are used to slow down running to a humanly understandable pace.

22/3/2022

SCC 110 L19

C-style solutions -

A function could return a sentinel value that indicates an error (eg -1 or null). But if:

* they are possible valid values
* the programmer forgets to check against sentinel value
* the programmer is writing a function for others to use and can't know what error correcting action to take each situation.

Unlike C, Java language won't allow invalid programs to run with invalid data.

Exceptions -

Java's way of handling error cases in your program. The exception is an (error) object which represents information about an error and changes the program control flow.

Exception Handler -

```
try {
    //Code which might cause an exception
} catch (AnExceptionClass exp) {
    //Code to run if exception happens
}
```

The variable declaration in the catch line is the class name (type) of the exception and the variable name for the associated exception object.

Happy and Sad paths -

Happy path is route through code where everything goes to plan.

Sad path is when something unexpected happens and your program needs to deal with them.

You can also add a single finally block after catch blocks which will always be run no matter how you leave the try block

Checked vs Unchecked Exceptions -

Checked: Must be caught or re-thrown
* Methods / Constructors must declare any checked exceptions they may cause but not handle
* Enforced by compiler
* Sub-classes of exception

Unchecked: (runtime) can be caught but don't have to be checked.
* Sub-classes of RuntimeException

Throws -

If you don't handle a checked exception inside a method you must declare the method to throw that exception. Code that calls that method must catch the exception or declare it might not deal with it as well.

You can create your own custom exceptions to represent exceptional conditions that are specific to the problem you are trying to solve.

22/3/2022

SCC 110 L20 Testing

Closed-box and open-box testing concepts -

Closed-Box -

Tests the program without looking at it's internal structure, derive the tests from the specification, aims to cover as much of the specification as possible.

Open-Box -

Test the program with knowledge of it's internal structure aiming to cover as many code paths as possible.

Equivalence partitioning / Boundary Testing -

Equivalence Partitioning -

Software often has categories of different inputs which produce different outputs. Test each equivalence category at least once.

Boundary Testing -

Look at the ranges of input values and the point at which the specified program behaviour should change. Can be used to identify equivalence partitions

Data Input Validation -

Creating boundaries and suitable / unsuitable inputs. Empty space etc.

Pre / Post Condition Tests -

Some properties of data should be the same before and after a method is called.

Unit Testing -

Unit testing tests individual units of functionality in your program such as methods against expected outputs.

JUnit is Unit Testing framework for Java

* Download junit and hamcrest-core jar files
* Add unit tests to program
* Compile

Windows: `javac -cp ".;junit-4.13.2.jar;hamcrest-core-1.3.jar" ArrayCalculate.java ArrayCalculateTests.java`

Linux: `javac -cp ".:junit-4.13.2.jar:hamcrest-core-1.3.jar" ArrayCalculate.java ArrayCalculateTests.java`

Run Tests

Windows: `java -cp ".;junit-4.13.2.jar;hamcrest-core-1.3.jar" org.junit.runner.JUnitCore ArrayCalculateTests`

Linux: `java -cp ".:junit-4.13.2.jar:hamcrest-core-1.3.jar" org.junit.runner.JUnitCore ArrayCalculateTests`

Demo:

```
import static org.junit.Assert.*;
import org.junit.Test;
public class ArrayCalculateTests {
    @Test
    public void normalTest() {
        int[] arrayIn = {1,2,3};
        int[] arrayOut = {2,4,6};
        ArrayCalculate c = new ArrayCalculate();
        assertArrayEquals(c.timesTwo(arrayIn), arrayOut);
    }
}
```

Pass the test to see if assertion is True. See org.junit.Assert

9/5/2022

SCC 110 L23 Continous Integration

Git workflow reminder

1) Clone Repo
2) Make changes with editor
3) Compile and test changes
4) git add all changed files in commit
5) git commit to finalise the change
6) git push to make changes available to others.
7) Go to 2.

Continous Integration -

The practice of automating the testing and merging of one or multiple developers contributions via a version control repository.

A Basic CI System -

- A server monitors the repository for commits
- When one is detected, clone the repository
- Build the program (auto-generate resources, compile, package etc.)
- Run all the tests
- Report back on the results

In this approach the version being tested is the current repo version.

Environment. -

To be able to build and test the CI system needs to run these programs somewhere. These are often called runners.

Github Actions -

Creates a template workflow
Actions Tab → Templates → Workflow file
YAML (text format based on indentation)
"on:" when workflow should be triggered
"jobs" how it runs
    "runs-on:" specifies OS image used
    "steps:" pre-prepared actions and commands

Continious Development (CD) -

Automatically deploy the system if the build and tests pass