Week-1

Experiment No: 1

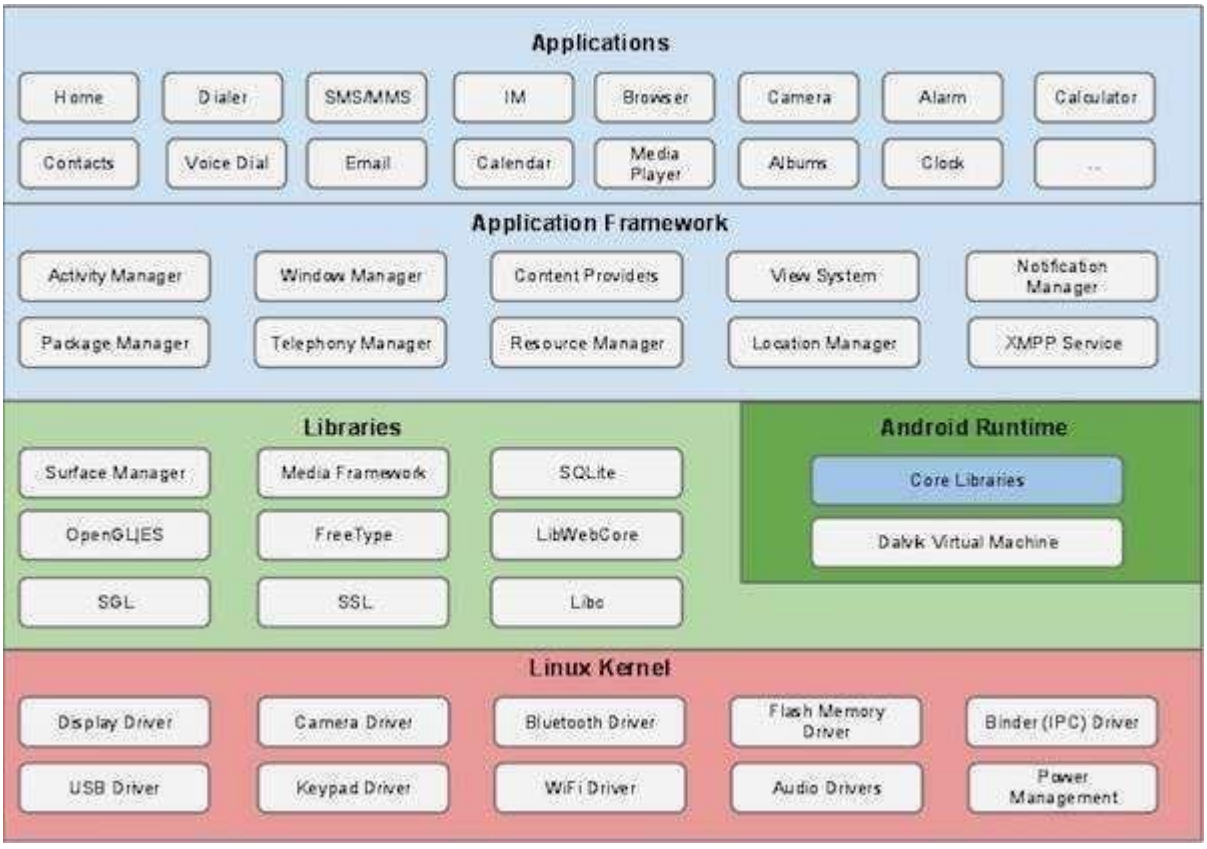
Module 1:

Android OS Architecture: Application Layer, Framework Layer, Libraries and Runtime, Hardware Abstraction Layer, and Kernel

Task: Select any two Mobile Apps used in your mobile phone and note the various functionalities and their corresponding layers

Android OS Architecture:

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as

networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.
- **android.database** − Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to render and manipulate text on a device display.
- **android.view** − The fundamental building blocks of application user interfaces.
- **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services −

- **Activity Manager** − Controls all aspects of the application lifecycle and activity stack.
- **Content Providers** − Allows applications to publish and share data with other applications.
- **Resource Manager** − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** − Allows applications to display alerts and notifications to the user.
- **View System** − An extensible set of views used to create application user interfaces.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

Task: Select any two Mobile Apps used in your mobile phone and note the various functionalities and their corresponding layers

Face book functionalities:

- **Timeline.** User profiles and updates are shown on what it known as the Timeline. Timeline is the successor to the Facebook wall, which was the original home for user profiles and updates. The user timeline includes posts, status updates, friend listings, photos, videos and user activity information.
- **Friends.** A primary feature of Facebook is the ability to search for and connect with friends and family. The search interface helps users quickly find acquaintances and also suggests potential connections.
- **News Feed.** News Feed enables users to view news from the connections and groups that they follow. Users can like a given post or comment on it.
- **Pages.** Pages are the profile and content pages for businesses on Facebook. Pages provide the ability for businesses to share information and communicate with customers.
- **Games.** Facebook provides an integrated capability enabling users to play games on their own or together with friends. Among the early successes of games on Facebook was Zynga's FarmVille.
- **Groups.** Communities of interest can organize themselves with the Facebook group feature. This enables the sharing of information, images and active discussions.
- **Events.** This feature enables users and groups to organize events that their followers can attend. It enables users to send out invites and help manage an attendee list.
- **Marketplace.** This is an online yard sale, where users can buy and sell goods and services with other Facebook members.
- **Messenger.** This is an instant messenger that enables friends to communicate in real time via webchat or a mobile app.
- **Video.** Facebook Live is a feature that enables individuals and businesses to stream live video to friends, family and followers.

What's app functionality:

1. Same WhatsApp, Multiple devices

After years of waiting, WhatsApp finally gave us the ability to use our same WhatsApp account on multiple devices. So, the feature basically works like how you use WhatsApp Web. All you need to do is — suppose you want to run WhatsApp on a different phone. Simply, download WhatsApp on the new device, and while you're on the screen that says enter your phone number, simply tap the three dots in the top-right and select Link to existing account. After that, scan the QR code from your primary phone, and voila, you can now run the same WhatsApp account on two smartphones. And you can simultaneously do this on 4 different devices, which is pretty cool and handy.

2. Chat lock

Chat Lock is another cool feature that has been a long time coming. Up until now, to hide chats, either you could archive them or lock WhatsApp altogether. But now, you can specifically lock WhatsApp chats. To do this, simply head over to the profile info screen of any chat contact that you'd like to lock. Then, scroll down to find the 'Chat lock' option. On the next screen, enable "Lock this chat with fingerprint", authenticate and you are done.

3. Edit messages

Similar to Telegram, you can now edit sent WhatsApp messages, basically to rectify any mistake or edit your message. To do this, tap and hold on to the message you wish to edit. Now select the 'Edit' option from the three-dots menu in the top-right. Make the changes to the text and hit the 'tick' option to finalise the changes. Note, you can only edit texts within the first 15 minutes, and there's going to be an edited tag below the edited message.

4. Share high-quality photos

Up until now, if you had to share a high-quality photo with someone on WhatsApp, you had to rely on sending a photo as a document. Well, not anymore. Simply, go to WhatsApp Settings, look for Storage and Data, and within Media Upload Quality, choose "Best quality" for Photo Upload quality.

This way, you can send your photos in the best possible quality on WhatsApp without relying on the "sending photo as document" feature.

5. Dedicated video recording mode

Previously, to directly record a video from WhatsApp, users had to press and hold the shutter button in the camera section of WhatsApp. But now, with the dedicated video recording mode, there's a separate button altogether that lets you record videos.

6. Voice status

We all know how to send voice messages on WhatsApp. But did you know you can now set voice messages as your status updates? Very simple. Head over to the 'Status' tab on WhatsApp and select the 'pencil' icon at the bottom. On the next screen, tap the 'microphone' icon and start recording your voice message for up to 30 seconds.

7. Status link previews

When posting a link as your status, WhatsApp can now automatically add a preview image by fetching the thumbnail or the featured image from the URL. Basically, with this feature, the person hitting your link will have context by seeing the thumbnail, like what the link is actually talking about.

Instagram

Instagram is one of the biggest social media platforms nowadays. It's almost impossible to have a successful social media marketing campaign without Instagram. As a social network built around photos and videos, Instagram made its way to popularity thanks to its simple filter feature that can make any photo look high-quality with little effort. But if the platform only had one feature, it wouldn't be where it is right now.

There are tons of Instagram features that are being updated constantly. In fact, it can be quite overwhelming trying to keep up with all of them. But if you want to make the most out of your Instagram, there's a few that you should focus on. So we're discussing the 10 most important Instagram features that you should be utilizing in your Instagram marketing strategy.

1. Instagram Live Video

What Is Live Video?

Live Video is a feature that enables Instagram users to stream video in real-time. Users can directly communicate with their followers and engage by comments and reactions.

While users broadcast a live video on their Instagram account, the profile photo, which is in the story section, has a light ring that notifies their followers to join the live broadcast.

Instagram Live Features

Instagram Live Video includes quite a lot of useful features that enable users to engage with followers. The noteworthy ones are:

- Broadcast video in real-time
- Save your live video to replay in Instagram Stories
- Followers can comment and react
- Pin comments to the top of the video
- Invite a friend who is watching the live video to stream together
- View engagement numbers after broadcasting

Week-2:

Module 2:

Android Studio: Install Android Studio, SDK Manager, Configure Plugins, Android Virtual Device(AVD) Emulators

Task: Install Android Studio and Configure Latest Android SDKs and Android Virtual Devices

Install Android Studio

Set up Android Studio in just a few clicks. First, check the system requirements. Then download the latest version of Android Studio.

Windows

Note: Windows machines with ARM-based CPUs aren't currently supported.

Here are the system requirements for Windows:

Requirement	Minimum	Recommended
OS	64-bit Microsoft Windows 8	Latest 64-bit version of Windows
RAM	8 GB RAM	16 GB RAM or more
CPU	x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor Framework.	Latest Intel Core processor
Disk space	8 GB (IDE and Android SDK and Emulator)	Solid state drive with 16 GB or more
Screen resolution	1280 x 800	1920 x 1080

To install Android Studio on Windows, follow these steps:

- If you downloaded an .exe file (recommended), double-click to launch it.
- If you downloaded a .zip file:

1. Unpack the .zip.
2. Copy the **android-studio** folder into your **Program Files** folder.
3. Open the **android-studio > bin** folder.
4. Launch studio64.exe (for 64-bit machines) or studio.exe (for 32-bit machines).
5. Follow the **Setup Wizard** in Android Studio and install any recommended SDK packages.

The following video shows each step of the setup procedure for the recommended .exe download:

As new tools and other APIs become available, Android Studio notifies you with a pop-up. To manually check for updates, click **Help > Check for Update**.

Mac

Here are the system requirements for Mac:

Requirement	Minimum	Recommended
OS	MacOS 10.14 (Mojave)	Latest version of MacOS
RAM	8 GB RAM	16 GB RAM or more
CPU	Apple M1 chip, or 2nd generation Intel Core or newer with support for Hypervisor Framework.	Latest Apple Silicon chip
Disk space	8 GB (IDE and Android SDK and Emulator)	Solid state drive with 16 GB or more
Screen resolution	1280 x 800	1920 x 1080

To install Android Studio on your Mac, follow these steps:

1. Launch the Android Studio DMG file.
2. Drag and drop Android Studio into the Applications folder, then launch Android Studio.
3. Choose whether to import previous Android Studio settings, then click **OK**.
4. Complete the Android Studio **Setup Wizard**, which includes downloading the Android SDK components that are required for development.
   As new tools and other APIs become available, Android Studio notifies you with a pop-up. To manually check for updates, click **Android Studio > Check for Updates**.

Linux

Note: Linux machines with ARM-based CPUs aren't currently supported.

Here are the system requirements for Linux:

Requirement	Minimum	Recommended
OS	Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE; GNU C Library (glibc) 2.31 or later.	Latest 64-bit version of Linux
RAM	8 GB RAM	16 GB RAM or more

CPU	x86_64 CPU architecture; 2nd generationIntel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3.	Latest Intel Core processor
Disk space	8 GB (IDE and Android SDK and Emulator)	Solid state drive with 16 GB or more
Screen resolution	1280 x 800	1920 x 1080

To install Android Studio on Linux, follow these steps:

1. Unpack the .zip file you downloaded to an appropriate location for your applications, such as within /usr/local/ for your user profile or /opt/ for shared users.
   For a 64-bit version of Linux, first install the required libraries for 64-bit machines.

2. To launch Android Studio, open a terminal, navigate to the android-studio/bin/ directory, and execute studio.sh.
3. Select whether you want to import previous Android Studio settings, then click **OK**.
4. Complete the Android Studio **Setup Wizard**, which includes downloading the Android SDK components that are required for development.

Task: Install Android Studio and Configure Latest AndroidSDKs and Android Virtual Devices

System Requirements

Before downloading and installing Android Studio, the following requirements are essential.

- Operating System Version - Microsoft Windows 7/8/10 (32-bit or 64-bit).
- Random Access Memory (RAM) - Minimum 4 GB RAM and 8 GB RAM recommended.
- Free Disk Space - Minimum 2 GB and 4 GB recommended.
- Minimum Required JDK Version - Java Development Kit (JDK) 8.
- Minimum Screen Resolution - 1280 * 800.resolution

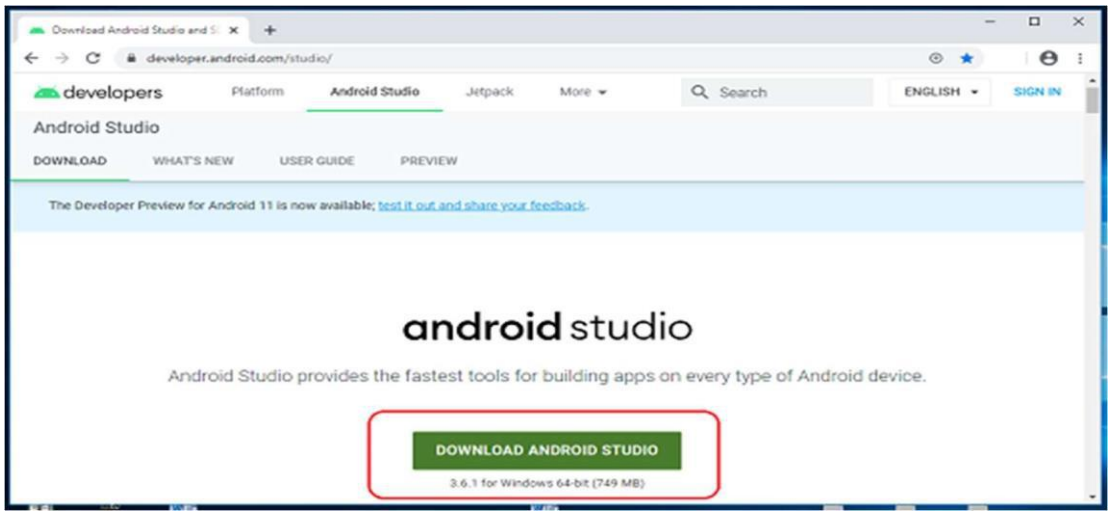
Download and Install Android Studio

Step 1

To download the Android Studio, visit the official Android Studio website in your web browser.

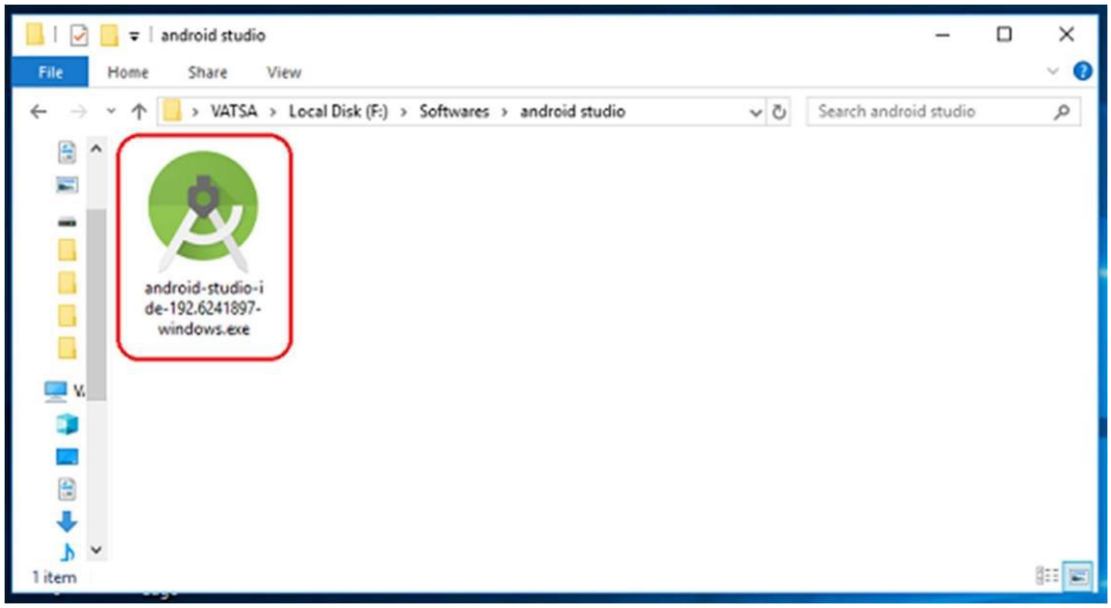
Step 2

Click on the "Download Android Studio" option.



Step 3

Double click on the downloaded "Android Studio-ide.exe" file.

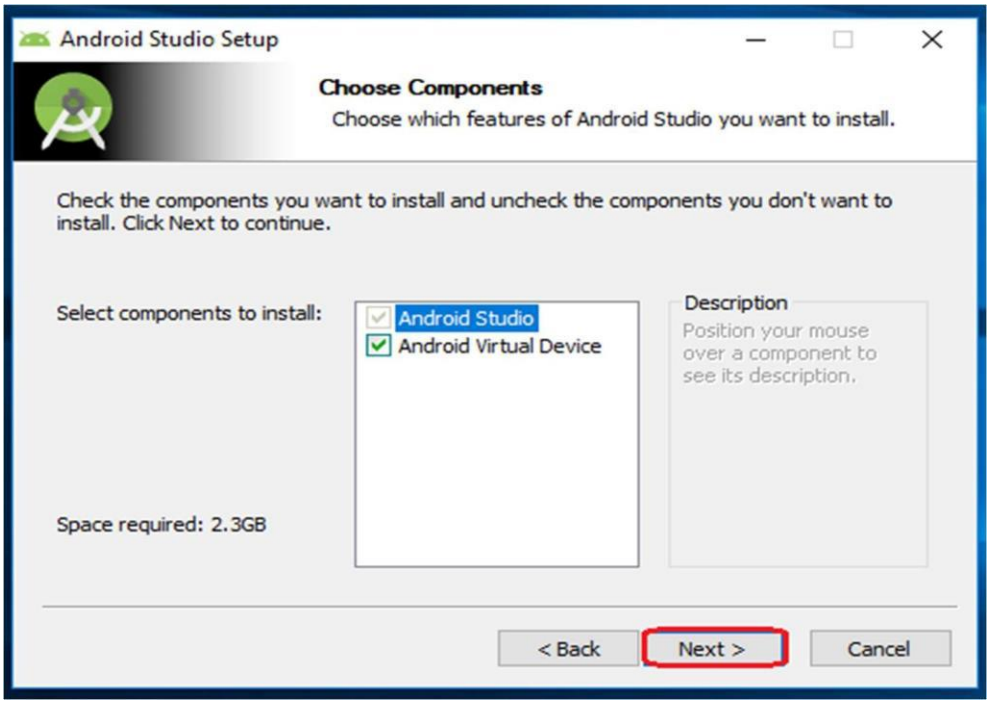


Step 4

"Android Studio Setup" will appear on the screen and click "Next" to proceed.

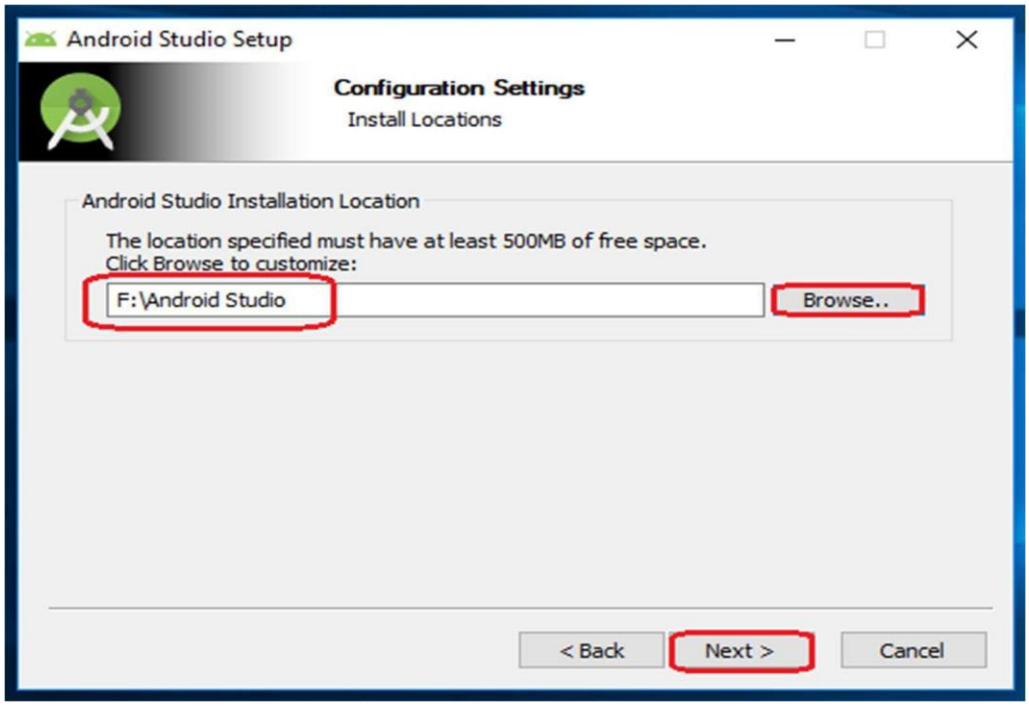
Step 5

Select the components that you want to install and click on the "Next" button.



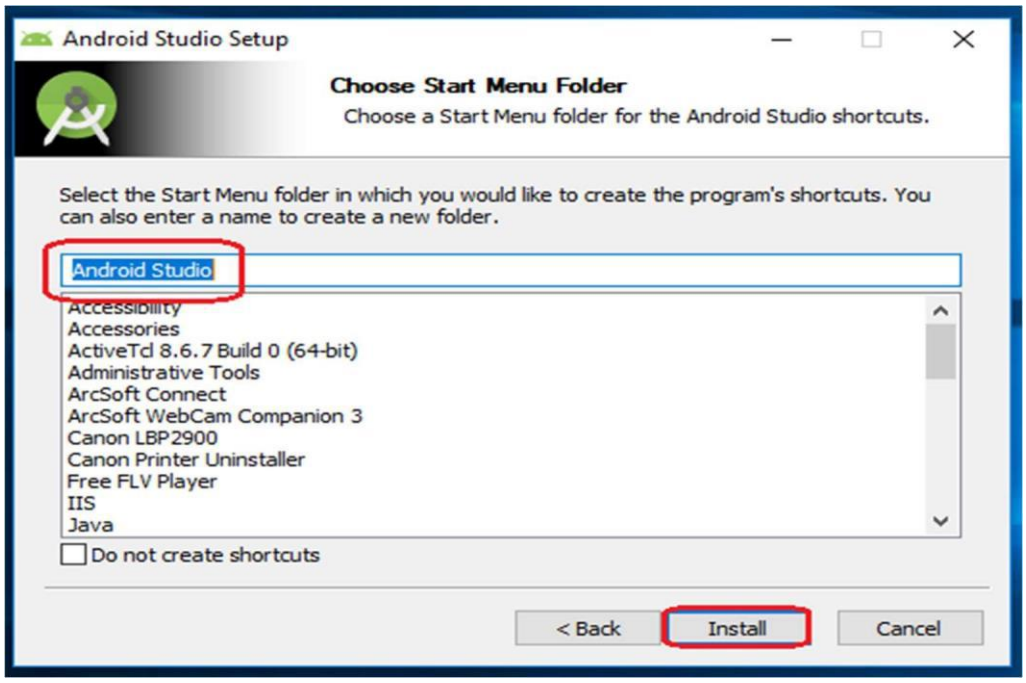
Step 6

Now, browse the location where you want to install the Android Studio and click "Next" to proceed.



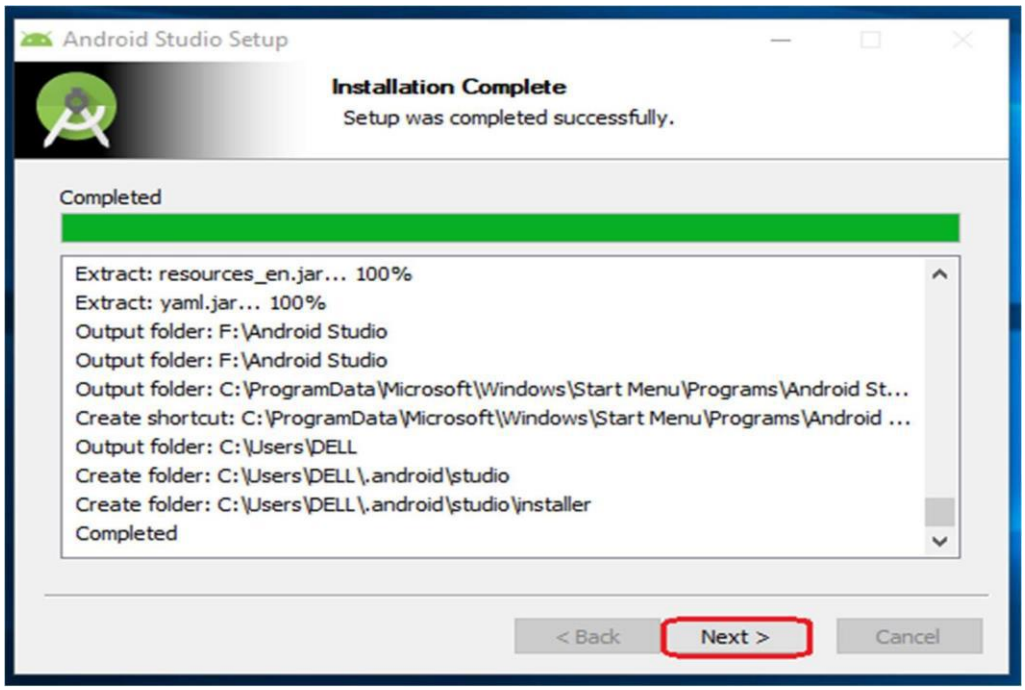
Step 7

Choose a start menu folder for the "Android Studio" shortcut and click the "Install" button to proceed.



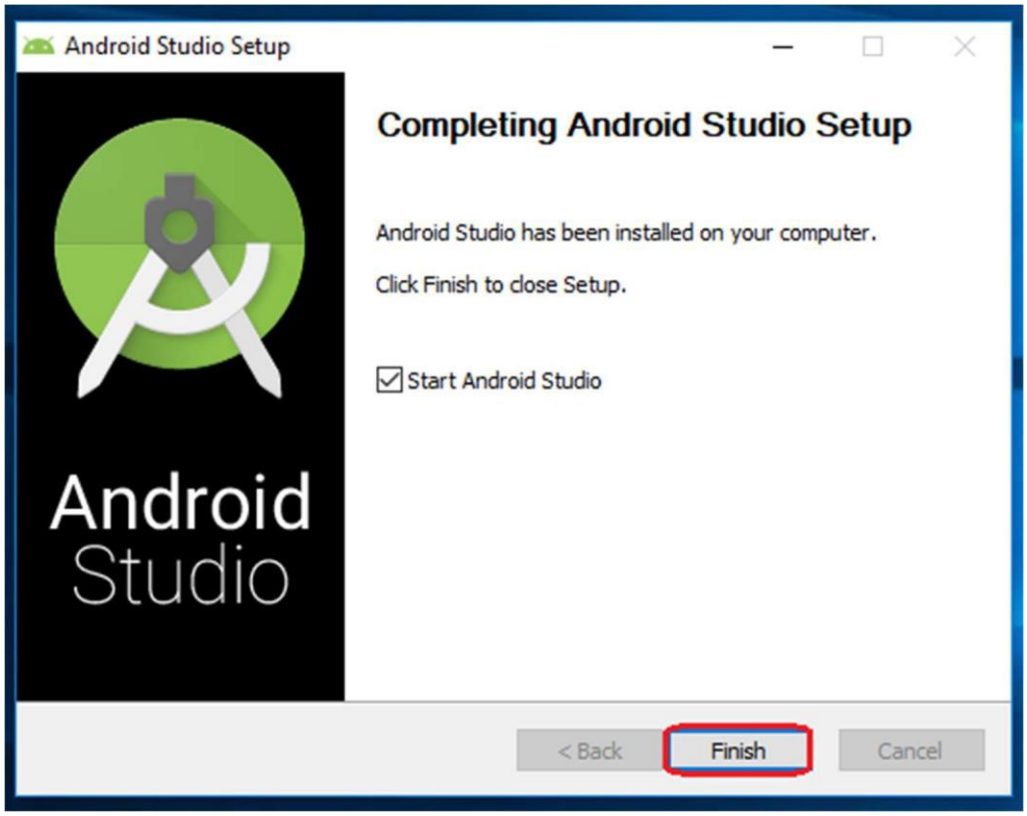
Step 8

After the successful completion of the installation, click on the "Next" button.

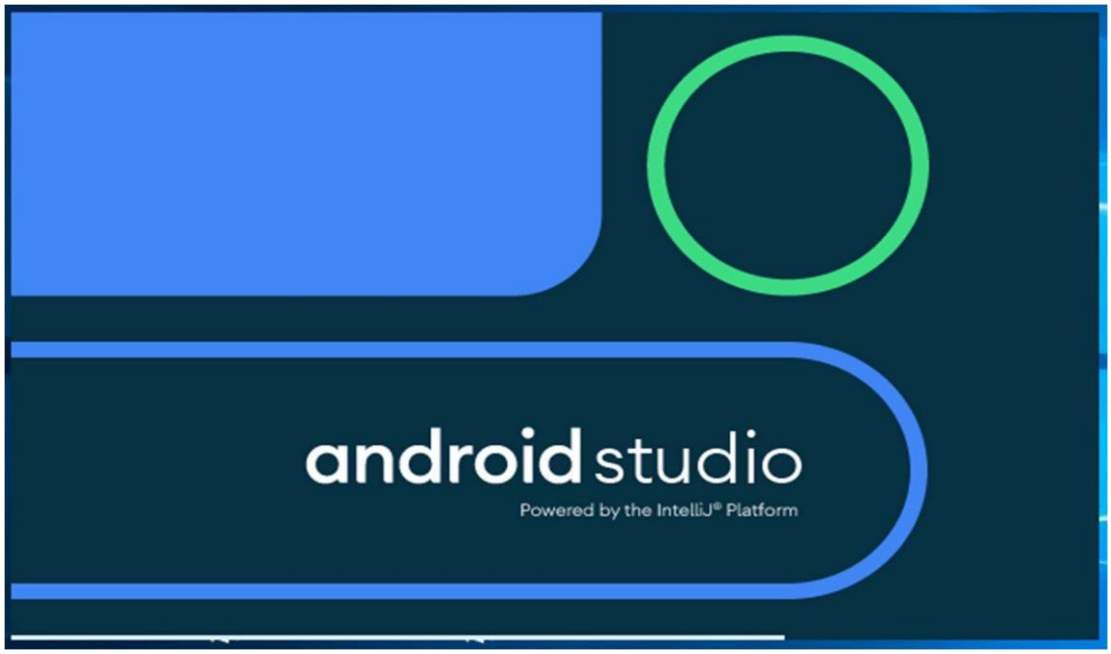


Step 9

Click on the "Finish" button to proceed.



Now, your Android studio welcome screen will appear on the screen.



Android Studio Setup Configuration

Step 10

"Android Studio Setup Wizard" will appear on the screen with the welcome wizard. Click on the "Next" button.

Step 11

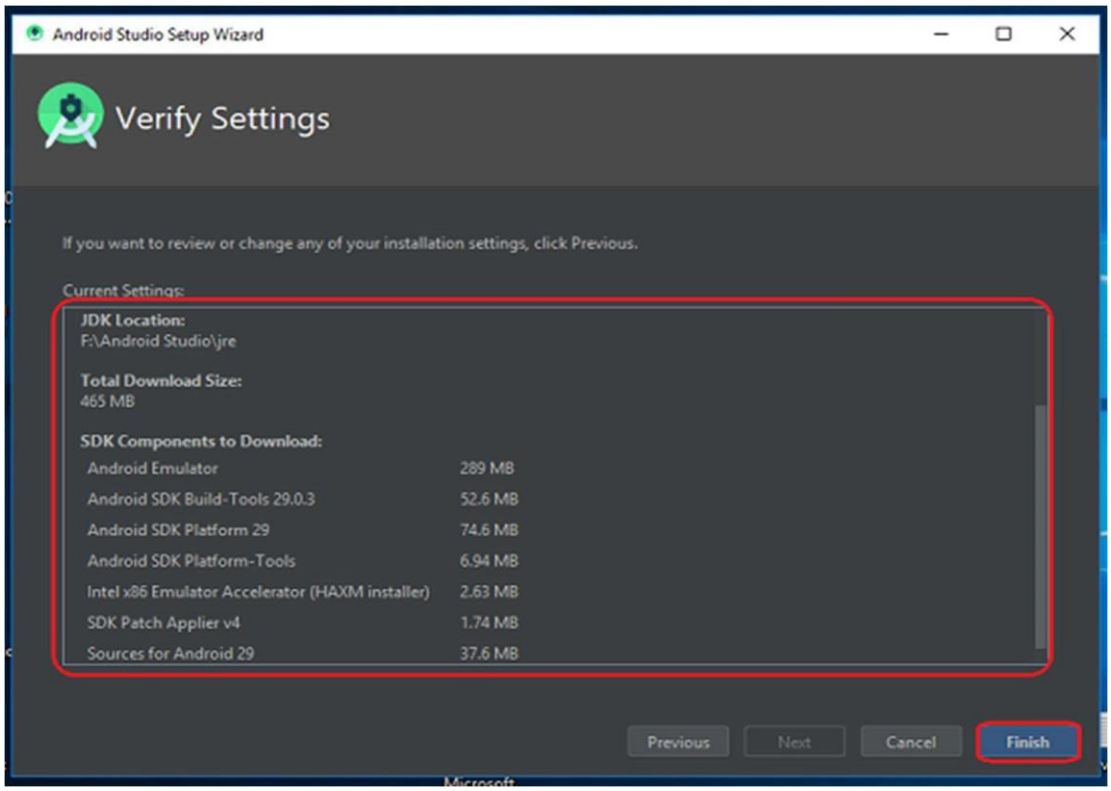
Select (check) the "Standard" option if you are a beginner and do not have any idea about Android Studio. It will install the most common settings and options for you. Click "Next" to proceed.

Step 12

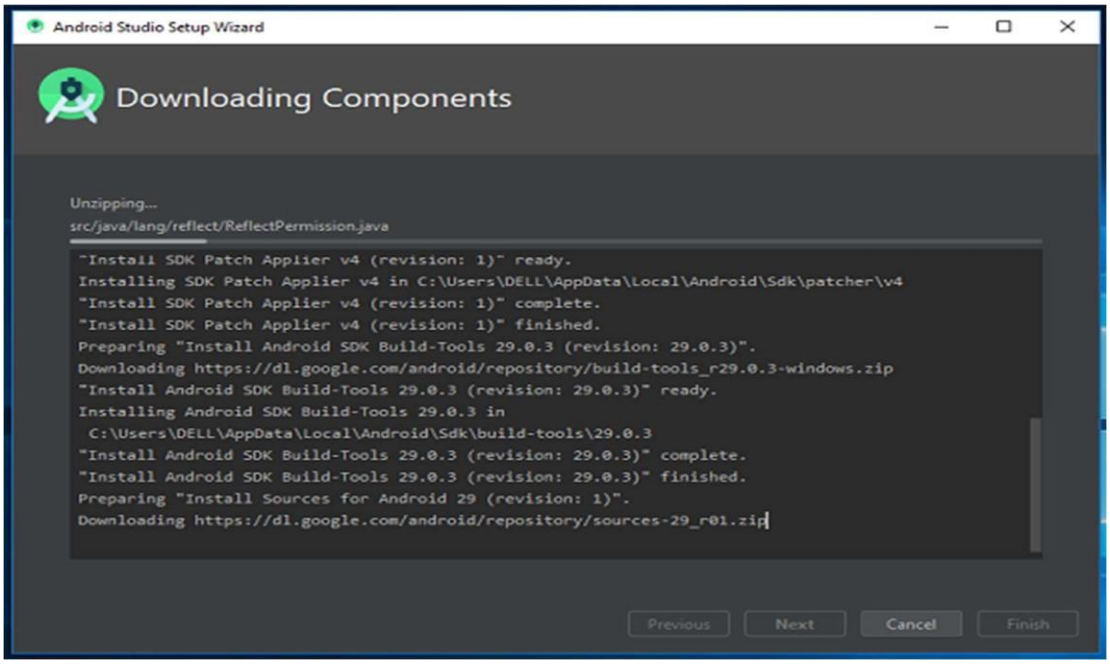
Now, select the user interface theme as you want. (I prefer Dark theme (Dracula) that is most liked by the coders). Then, click on the "Next" button.

Step 13

Now, click on the "Finish" button to download all the SDK components.

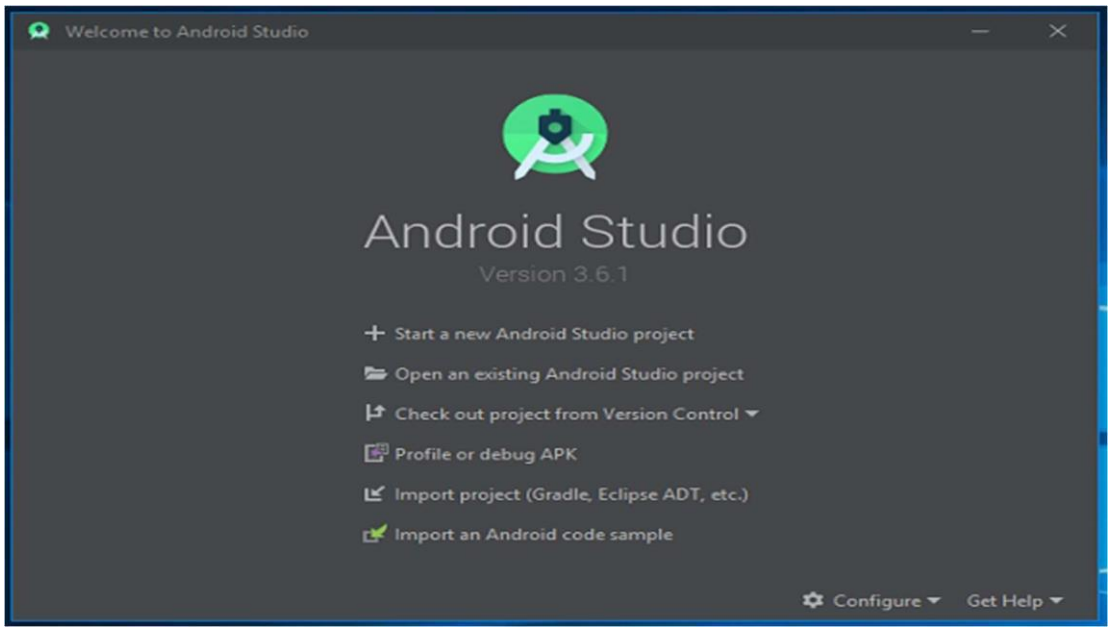


And, the downloading and installation process of components gets started.



Step 14

After downloading all the necessary components, click on the "Finish" button.



Congrats, your Android Studio has been successfully installed in your systemand you can start a new Android studio project.

Week-3

Module 3:

Building your First Application: Understanding Activities and Intents, Activity Lifecycle and Managing State, Activities and Implicit Intents

Task: Build and Run Hello World Application on the virtual Device and also test the app on your mobile phone.

About activities

An activity represents a single screen in your app with an interface the user can interact with. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading individual messages. Your app is a collection of activities that you either create yourself, or that you reuse from other apps.

Although the activities in your app work together to form a cohesive user experience in your app, each one is independent of the others. This enables your app to start activities in other apps, and other apps can start your activities (if your app allows it). For example, a messaging app you write could start an activity in a camera app to take a picture, and then start the activity in an email app to let the user share that picture in email.

Typically, one activity in an app is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start other activities in order to perform different actions.

Android Intent

Android Intent is the message that is passed between components such as activities, content providers, broadcast receivers, services etc.

It is generally used with startActivity() method to invoke activity, broadcast receivers etc.

The dictionary meaning of intent is intention or purpose. So, it can be described as the intention to do action.

The LabeledIntent is the subclass of android.content.Intent class.

Android intents are mainly used to:

- **Start the service**
- **Launch an activity**
- **Display a web page**
- **Display a list of contacts**
- **Broadcast a message**
- **Dial a phone call etc.**

Types of Android Intents

There are two types of intents in android: implicit and explicit.

1) Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

```
1. Intent intent=new Intent(Intent.ACTION_VIEW);
2. intent.setData(Uri.parse("http://www.javatpoint.com"));
3. startActivity(intent);
```

2) Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

```
1. Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
2. startActivity(i);
```

To get the full code of explicit intent, visit the next page.

Task:Build and Run Hello World Application on the virtual Device and also test the app on your mobile phone.

Step-1:Visit the site https://www.mobiroller.com/en/

Step-2:Select "Try it for free" button

Step-3: Select "Crate an account" Button

a. Enter E-Mail Address

b. Enter full name

c. Enter Password

After entering the above details Press "Create New Account".

Step-4: Login the page

Step-5: Select "+ Create New App"

Step-6: Enter App Name

Step-7: Select Any "Template".

Step-8: Press the button "Go to Control Panel"

After Entering Control Panel, we will observe our app completed 80% will Displayed.

(if you want to Change Template, change it)

Step-9:Select Content Button.

Select Modules (if Any) (I am Selected Website Module)

Enter Title and Websites link.

Select Save Option Button.

Step-10: Select "Add Module Button".

Select Modules (if Any) (I am Selected Standard Content Module)

Enter Title Name

Enter Context name

(if you want to add any image , add it)

Select Save Option Button.

Step-11: Select "Appearance "Button

Go to General settings & if you want to change font color, Animation

Select Save Option Button.

Step-12: Press "Generate APK" button

Press "Yes" button

Press "generate APK"

After generating Apk link we will receive a mail.

Finally, copy the link and install it in your mobile phone.

Week-4

Experiment No: 4

Module 4: Android UI components: Text Controls, Buttons, Widgets, Layouts, Containers Task: Explore all the UI Controls and design a Student Registration Activity Android UI Controls There are number of UI controls provided by Android that allow you to build the graphical user interface for your app. Sr.No. UI Control & Description 1 TextView This control is used to display text to the user. 2 EditText EditText is a predefined subclass of TextView that includes rich editing capabilities. 3 AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing. 4 Button A push-button that can be pressed, or clicked, by the user to perform an action. 5 ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user. 6 CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive. 7 ToggleButton An on/off button with a light indicator. 8 RadioButton The RadioButton has two states: either checked or unchecked. 9 RadioGroup A RadioGroup is used to group together one or more RadioButtons. 10 ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background. 11 Spinner A drop-down list that allows users to select one value from a set. 12 TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. 13 DatePicker The DatePicker view enables users to select a date of the day. Android UI Layouts Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. Generally, every application is a combination of View and ViewGroup. As we know, an android application contains a large number of activities and we can say each activity is one page of the application. So, each activity contains multiple user interface components and those components are the instances of the View and ViewGroup. All the elements in a layout are built using a hierarchy of View and ViewGroup objects. View A View is defined as the user interface which is used to create interactive UI components such as TextView, ImageView, EditText, RadioButton, etc., and is responsible for event handling and drawing. They are Generally Called Widgets. View A ViewGroup act as a base class for layouts and layouts parameters that hold other Views or ViewGroups and to define the layout properties. They are Generally Called layouts. ViewGroup The Android framework will allow us to use UI elements or widgets in two ways:• Use UI elements in the XML file • Create elements in the Kotlin file dynamically Types of Android Layout • Android Linear Layout: LinearLayout is a ViewGroup subclass, used to provide child View elements one by one either in a particular direction either horizontally or vertically based on the orientation property. • Android Relative Layout: RelativeLayout is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the

parent). • Android Constraint Layout: ConstraintLayout is a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but having more power. • Android Frame Layout: FrameLayout is a ViewGroup subclass, used to specify the position of View elements it contains on the top of each other to display only a single View inside the FrameLayout. • Android Table Layout: TableLayout is a ViewGroup subclass, used to display the child View elements in rows and columns. • Android Web View: WebView is a browser that is used to display the web pages in our activity layout. • Android ListView: ListView is a ViewGroup, used to display scrollable lists of items in a single column. • Android Grid View: GridView is a ViewGroup that is used to display a scrollable list of items in a grid view of rows and columns. Android Login and Registration Screen Design Create a new android application using android studio and give names as LoginExample. In case if you are not aware of creating an app in android studio check this article Android Hello World App. Once we create an application, open activity_main.xml file from \res\layout folder path and write the code like as shown below. activity_main.xml Now we will create another layout resource file registration.xml in \res\layout path to allow new users to register in our application for that right click on your layout folder à Go to New à select Layout Resource File and give name as registration.xml. Once we create a new layout resource file registration.xml, open it and write the code like as shown below registration.xml Now open your main activity file MainActivity.java from \java\com.tutlane.loginexample path and write the code like as shown below MainActivity.java package com.tutlane.loginexample; import android.content.Intent; import android.support.v7.app.AppCompatActivity; import android.os.Bundle; import android.text.method.LinkMovementMethod; import android.view.View; import android.widget.TextView; public class MainActivity extends AppCompatActivity { @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity_main); TextView register = (TextView)findViewById(R.id.lnkRegister); register.setMovementMethod(LinkMovementMethod.getInstance()); register.setOnClickListener(new View.OnClickListener() { @Override public void onClick(View v) { Intent intent = new Intent(MainActivity.this, RegistrationActivity.cla ss); startActivity(intent); } }); } } If you observe the above code, whenever the user click on register link, we are redirecting the user from login screen to registration screen using “RegistrationActivity” for that create another activity file RegistrationActivity.java in \java\com.tutlane.loginexa mple path. Output : When we run the above example in the android emulator we will get a result like as shown

Week-5

1) **Aim: Write a C-Program to perform various arithmetic operations on pointer variables**
2) **Aim: Write a C-Program to demonstrate the following parameter passing mechanisms:**
   - **i) call-by-value**
   - **ii) call-by-reference**

Source Code:

call-by-value

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b);
// printing the value of a and b in main
   swap(a,b);
```

```
    printf("After swapping values in main a = %d, b = %d\n",a,b);
// The value of actual parameters do not change by changing the formal parameters in c
all by value, a = 10, b = 20
}
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b);
// Formal parameters, a = 20, b = 10
}
```

Input & Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

call-by-reference

Source Code:

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the valu
e of a and b in main
    swap(&a,&b);
```

```
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual p
arameters do change in call by reference, a = 10, b = 20
}
void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal paramete
rs, a = 20, b = 10
}
```

Input & Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

Week-6

1) **Aim: Write a C-program that uses functions to perform the following operations:**
   i) Reading a complex number

ii) Writing a complex number
iii) Addition of two complex numbers
iv) Multiplication of two complex numbers

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct
{
        float x;
        float y;
}complex;
void main()
{
        complex cn1,cn2,sum,product;
        complex read(void);
        void display(complex);
        complex add(complex ,complex);
        complex multiply(complex ,complex);
        cn1=read();
        cn2=read();
        printf("\n two complex numbers:");
        display(cn1);
```

```
        display(cn2);
        sum=add(cn1,cn2);
        printf("\n sum of these two numbers:");
        display(sum);
        product =multiply(cn1,cn2);
        printf("\n product of these two numbers:");
        display(product);
}
complex read()
{
        complex cn;
        printf("\n input a complex number:");
        printf("\n real part:");
        scanf("%f",&cn.x);
        printf("\n imaginary part:");
        scanf("%f",&cn.y);
        return cn;
}
void display(complex cn)
{
        printf("\n %6.2f +i %6.2f",cn.x,cn.y);

}
```

```
complex add(complex cn1,complex cn2)
{
        complex cn;
        cn.x=cn1.x+cn2.x;
        cn.y=cn1.y+cn2.y;
        return cn;
}
complex multiply(complex cn1,complex cn2)
{
        complex cn;
        cn.x=cn1.x*cn2.x-cn1.y*cn2.y;
        cn.y=cn1.x*cn2.y+cn2.x*cn1.y;
        return cn;
}
```

Input & Output:

```
 input a complex number:
 real part:5
 imaginary part:3
 input a complex number:
 real part:8
 imaginary part:4
 two complex numbers:
```

5.00 +i 3.00

8.00 +i 4.00

sum of these two numbers:

13.00 +i 7.00

product of these two numbers:

28.00 +i 44.00

Week-7

1)Aim: Write C-programs that implement stack (its operations) using

i) Arrays
ii) Pointers

Source Code:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 6

int stack[MAX];

int top=0;

int menu()
```

```
{
        int ch;
        printf("\n ..stack operations using array");
        printf("\n.....*******...\n");
        printf("\n 1.push");
        printf("\n 2.pop");
        printf("\n 3.display");
        printf("\n 4.quit");
        printf("\n Enter your choice");
        scanf("%d",&ch);
        return ch;
}
void display()
{
        int i;
        if(top==0)
        {
                printf("\n\n stack empty");
                return;
        }
        else
        {
                printf("\n\n Elements in stack:");
```

```
            for(i=0;i<top;i++)
                printf("\t%d",stack[i]);
        }
}
void pop()
{
        if(top==0)
        {
            printf("\n\n Stack underflow...");
            return;
        }
        else
        {
            printf("\n\n popped element is:%d",stack[--top]);
        }
}
void push()
{
        int data;
        if(top==MAX)
        {
            printf("\n\n stack overflow..");
            return;
```

```
        }
        else
        {
                printf("\n\n Enter data:");
                scanf("%d",&data);
                stack[top]=data;
                top=top+1;
                printf("\n\n Data pushed into the stack:");
        }
}
void main()
{
int ch;
do{
        ch=menu();
        switch(ch)
        {
        case 1:push();
        break;
        case 2:pop();
        break;
        case 3:display();
        break;
```

```
        case 4:exit(0);
            break;
        default:printf("\n Enter input is worng");
        }
}while(1);
}
```

Input & Output:

```
 ..stack operations using array
.....*******...
 1.push
 2.pop
 3.display
 4.quit
 Enter your choice1
 Enter data:12
 Data pushed into the stack:
 ..stack operations using array
.....*******...
 1.push
 2.pop
 3.display
 4.quit
```

Enter your choice1

Enter data:13

Data pushed into the stack:

..stack operations using array

.....*******...

1.push

2.pop

3.display

4.quit

Enter your choice1

Enter data:14

Data pushed into the stack:

..stack operations using array

.....*******...

1.push

2.pop

3.display

4.quit

Enter your choice3

Elements in stack: 12 13 14

..stack operations using array

.....*******...

1.push

2.pop

3.display

4.quit

Enter your choice2

popped element is:14

..stack operations using array

.....*******...

1.push

2.pop

3.display

4.quit

Enter your choice3

Elements in stack: 12 13

7a) Aim: Write a C-program to find stack linked list?

Sorce Code:

```
#include <stdio.h>
#include <stdlib.h>
 struct stack
 {
        int data;
        struct stack *next;
 };
 void push();
```

```
void pop();
void display();
typedef struct stack node;
node *start=NULL;
node *top=NULL;
node *getnode()
{
        struct stack *temp;
        temp=(node *)malloc(sizeof(node));
        printf("\n Enter data ");
        scanf("%d",&temp->data);
        temp->next=NULL;
        return temp;
}
void push(node*newnode)
{
        node *temp;
        if(newnode==NULL)
        {
                printf("\n stack overflow");
                return;
        }
        if(start==NULL)
```

```
        {
                start=newnode;
                top=newnode;
        }
        else
        {
                temp=start;
                while(temp->next!=NULL)
                        temp=temp->next;
                        temp->next=newnode;
                        top=newnode;
        }
        printf("\n Data pushed into stack");
}
void pop()
{
        node *temp;
        if(top==NULL)
        {
                printf("\n stack underflow");
                return;
        }
        temp=start;
```

```
        if(start->next==NULL)
        {
                printf("\n poped element is %d",top->data);
                start=NULL;
                free(top);
                top=NULL;
        }
        else
        {
                while(temp->next!=top)
                {
                        temp=temp->next;
                }
                temp->next=NULL;
                printf("\n poped element is %d",top->data);
                free(top);
                top=temp;
        }
}
 void display()
{
        node *temp;
        if(top==NULL)
```

```
        {
                printf("\n stack is empty");
        }
        else
        {
                temp=start;
                printf("\n element in the stack ");
                printf(" %5d",temp->data);
                while(temp!=top)
                {
                        temp=temp->next;
                        printf("%4d",temp->data);
                         }
        }
}
 int menu()
{
        int ch;
        printf("\n stack operations using pointers");
        printf("\n ------------------------------");
        printf("\n 1.push");
        printf("\n 2.pop");
        printf("\n 3.Display");
```

```
        printf("\n 4.Quit");
        printf("\n Enter your choice");
        scanf("%d",&ch);
        return ch;
 }
int main(void)
{
        int ch;
        node *newnode;
        do
        {
                ch=menu();
                switch(ch)
                {
                case 1: newnode=getnode();
                                push(newnode);
                                break;
                case 2:pop();
                                break;
                case 3:display();
                                break;
                case 4:exit(0);
                }
```

```
        }while(1);
}
```

Input & Output:

```
stack operations using pointers
 ------------------------------
 1.push
 2.pop
 3.Display
 4.Quit
 Enter your choice1
 Enter data 4
 Data pushed into stack
 stack operations using pointers
 ------------------------------
 1.push
 2.pop
 3.Display
 4.Quit
 Enter your choice1
 Enter data 5
 Data pushed into stack
 stack operations using pointers
```

```
------------------------------
1.push
2.pop
3.Display
4.Quit
Enter your choice3
element in the stack      4   5
stack operations using pointers
------------------------------
1.push
2.pop
3.Display
4.Quit
Enter your choice2
poped element is 5
stack operations using pointers
------------------------------
1.push
2.pop
3.Display
4.Quit
Enter your choice3
element in the stack      4
```

Week-8

1)Aim: Write C-programs that implement Queue (its operations) using

- **i) Arrays**
- **ii) Pointers**

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 6
int q[MAX];
int front,rear;
int menu()
{
        int ch;
        printf("\n ..Queue operations using array");
        printf("\n.....*******...\n");
        printf("\n 1.Insert");
        printf("\n 2.Delete");
        printf("\n 3.display");
        printf("\n 4.quit");
        printf("\n Enter your choice");
        scanf("%d",&ch);
        return ch;
}
void display()
{
        int i;
        if(front==rear)
```

```
        {
                printf("\n\n Queue is empty");
                return;
        }
        else
        {
                printf("\n\n Elements in Queue are:");
                for(i=front;i<rear;i++)
                        printf("\t%d",q[i]);
        }
}
void delete()
{
        if(rear==front)
        {
                printf("\n\n Queue is empty...");
                return;
        }
        else
        {
                printf("\n\n Deleted element is from queue:%d",q[front]);
                front++;
        }
```

```
}
void insert()
{
        int data;
        if(rear==MAX)
        {
                printf("\n\n Linear Queue is full..");
                return;
        }
        else
        {
                printf("\n\n Enter data:");
                scanf("%d",&data);
                q[rear]=data;
                rear=rear+1;
                printf("\n\n Data inserted into the queue:");

        }
}
void main()
{
int ch;
do{
```

```
        ch=menu();
        switch(ch)
        {
        case 1:insert();
        break;
        case 2:delete();
        break;
        case 3:display();
        break;
        case 4:exit(0);
              break;
        default:printf("\n Enter input is wrong");
        }
}while(1);
}
```

Input & Output:

```
            ..Queue operations using array
.....*******...
 1.Insert
 2.Delete
 3.display
 4.quit
```

Enter your choice:1

Enter data:4

Enter data:5

Enter data:6

..Queue operations using array

.....*******...

1.Insert

2.Delete

3.display

4.quit

Enter your choice:3

Elements in Queue are

4 5 6

8(a)Aim: Write a C-Program to find Queue using linked list?

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
 struct queue
{
        int data;
        struct queue *next;
};
```

```
typedef struct queue node;
node *front=NULL;
node *rear=NULL;
node *getnode()
{
        node *temp;
        temp=(node *)malloc(sizeof(node));
        printf("\n Enter data ");
        scanf("%d",&temp->data);
        temp->next=NULL;
        return temp;
}
void insertQ()
{
        node *newnode;
        newnode=getnode();
        if(newnode==NULL)
        {
                printf("\n queue full");
                return;
        }
        if(front==NULL)
        {
```

```
                front=newnode;
                rear=newnode;
        }
        else
        {
                        rear->next=newnode;
                        rear=newnode;
        }
        printf("\n Data inserted into the queue");
}
void deleteQ()
{
        node *temp;
        if(front==NULL)
        {
                printf("\n \n empty queue");
                return;
        }
        temp=front;
        front=front->next;
        printf("\n\n deleted element from queue is%d",temp->data);
        free(temp);
}
```

```
void displayQ()
{
        node *temp;
        if(front==NULL)
        {
                printf("\n\n\t empty queue");
        }
        else
        {
                temp=front;
                printf("\n\n elements in the queue are:");
                while(temp!=NULL)
                {
                printf("%4d",temp->data);
                temp=temp->next;
               }
        }
}
int menu()
{
        int ch;
        printf("\n queue operations using pointers");
        printf("\n ------------------------------");
```

```
        printf("\n 1.Insert");
        printf("\n 2.Delete");
        printf("\n 3.Display");
        printf("\n 4.Quit");
        printf("\n Enter your choice");
        scanf("%d",&ch);
        return ch;
 }
int main(void)
{
        int ch;

        do
        {
                ch=menu();
                switch(ch)
                {
                case 1: insertQ();
                                break;
                case 2:deleteQ();
                                break;
                case 3:displayQ();
                                break;
```

```
            case 4:exit(0);
            }
        }while(1);
}
```

Input & Output:

```
queue operations using pointers
------------------------------
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice1
Enter data 4
Data inserted into the queue
queue operations using pointers
------------------------------
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice1
Enter data 5
```

Data inserted into the queue

queue operations using pointers

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice2

deleted element from queue is4

queue operations using pointers

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice3

elements in the queue are: 5

Week-9

9)Aim: Write a C-program that uses Stack operations to perform the following:

i) **Converting infix expression into postfix expression**
ii) **Evaluating the postfix expression**

9)a)Aim: Write a C-Program infix to postfix conversion?

Source Code:

```
#include <stdio.h>
#include<string.h>
        char postfix[50];
        char infix[50];
        char opstack[50];
        int i,j,top=0;
       int lesspriority(char op,char op_at_stack)
        {
          int k;
          int pv1;
          int pv2;
          char operators[    ]={'+','-','*','/','%','^','('};
          int priority_value[      ]={0,0,1,1,2,3,4};
          if(op_at_stack=='(')
                  return 0;
          for(k=0;k<6;k++)
          {
                  if(op==operators[k])
                          pv1=priority_value[k];
                  }
          for(k=0;k<6;k++)
          {
                  if(op_at_stack==operators[k])
```

```
                pv2=priority_value[k];
    }
    if(pv1<pv2)
            return 1;
    else
            return 0;
}
void push(char op)
{
    if(top==0)
    {
            opstack[top]=op;
            top++;
    }
    else
    {
            if(op!='(')
            {
                    while(lesspriority(op,opstack[top-1])==1&&top>0)
                    {
                            postfix[j]=opstack[--top];
                            j++;
                    }
```

```
            }
            opstack[top]=op;
            top++;
  }
}
pop()
{
  while(opstack[--top]!='(')
  {
            postfix[j]=opstack[top];
            j++;
            }
}
void main()
{
  char ch;
  printf("\n enter infix expression");
 scanf("%s",infix);
 while((ch=infix[i++])!=0 )
 {
           switch( ch )
           {
            case '  ':   break;
```

```
            case '(':
            case '+':
            case '-':
            case '*':
           case '/':
            case '^':
            case '%':
                        push(ch);
                       break;
            case ')':
                              pop();
                        break;
            default :
                             postfix[ j ]= ch;
                        j++;
                        }
}
while(top>=0)
{
            postfix[j]=opstack[--top];
            j++;
}
postfix[j]='\0';
```

```
        printf("\n infix expression :%s",infix);
        printf("\n post fix expression :%s ",postfix);
       }
```

Input & Output:

```
enter infix expression(A+B)*C
infix expression :(A+B)*C
post fix expression :AB+C*
```

9)b)Aim: Write a C-Program for Postfix Expression Evaluation using Stack?

Source Code:

```
#include <stdio.h>
#include <math.h>
#include<ctype.h>
#include<string.h>
#define MAX 20
int isoperator(char ch)
{
        if(ch=='+'||ch=='*'||ch=='/'||ch=='^')
                return 1;
        else
```

```
        return 0;
}
void main()
{
    char postfix[MAX];

    char ch;
    int i=0,top=0;
    float val_stack[MAX],val1,val2,res;
    printf("\n Enter a post fix expression::");
    scanf("%s",postfix);
    while((ch=postfix[i])!= '\0')
    {
        if((isoperator(ch))==1)
        {
            val2=val_stack[--top];
            val1=val_stack[--top];
            switch(ch)
            {
            case'+':
                res=val1+val2;
                break;
            case'-':
```

```
                    res=val1-val2;
                    break;
                case'*':
                    res=val1*val2;
                    break;
                case'/':
                    res=val1/val2;
                    break;
                case'^':
                    res=pow(val1,val2);
                    break;
                }
                val_stack[top]=res;
            }
            else
            {
                val_stack[top]=ch-48;
            }
            top++;
            i++;
        }
        printf("\n values of %s is:%f",postfix,val_stack[0]);
}
```

Input & Output:

1. Enter a post fix expression::23+5*

   values of 23+5* is:25.000000

2. Enter a post fix expression::23+5/

   values of 23+5/ is:1.000000

Week-10

1) Aim: Write a C program that uses functions to perform the following operations on singly linked list.

i) Creation
ii) Insertion
iii) Deletion
iv) Traversal

```
#include <stdio.h>

#include <stdlib.h>

struct slinklist

{

int data;

struct slinklist *next;

};

typedef struct slinklist node;

node *start=NULL;

int menu()

{

        int ch;

        printf("\n 1.create a list");

        printf("\n----------------------");

        printf("\n 2.insert a node at beginning");

        printf("\n 3.insert a node at end");
```

```
        printf("\n 4.insert a node at middle");
        printf("\n-----------------------------");
        printf("\n 5.delete a node from beginning");
        printf("\n 6.delete a node from list");
        printf("\n 7.delete a node at middle");
        printf("\n-------------------------------");
        printf("\n 8.traverse the list(left to right)");
        printf("\n 9. traverse the list(right to left)");
        printf("\n-------------------------------------");
        printf("\n 10.count nodes");
        printf("\n 11.exit");
        printf("\n \n enter your choice:");
        scanf("%d",&ch);
        return ch;
}
node *getnode()
{
        node *newnode;
        newnode=(node*)malloc(sizeof(node));
        printf("\n enter data:");
        scanf("%d",&newnode->data);
        newnode->next=NULL;
        return newnode;
```

```
}
void createlist(int n)
{
        int i;
        node *newnode;
        node *temp;
        for(i=0;i<n;i++)
        {
                newnode=getnode();
                if(start==NULL)
                {
                        start=newnode;
                }
                else
                {
                        temp=start;
                        while(temp->next!=NULL)
                                temp=temp->next;
                        temp->next=newnode;
                }
        }
}
void insert_at_beg()
```

```
{
        node *newnode;
        newnode=getnode();
        if(start==NULL)
        {
                start=newnode;
        }
        else
        {
                newnode->next=start;
                start=newnode;
        }
}
void insert_at_end()
{
        node *newnode,*temp;
        newnode=getnode();
        if(start==NULL)
        {
                start=newnode;
        }
        else
        {
```

```
                temp=start;
                while(temp->next!=NULL)
                        temp=temp->next;
        }
}
void insert_at_mid()
{
        node *newnode,*temp,*prev;
        int i,pos,nodectr;
        newnode=getnode();
        printf("\n enter the position:");
        scanf("%d",&pos);
        nodectr=countnode(start);
        if(pos>1&&pos<nodectr)
        {
                temp=prev=start;
                i=1;
                while(i<pos)
                {
                        prev=temp;
                        temp=temp->next;
                        i++;
                }
```

```
            prev->next=newnode;
            newnode->next=temp;
        }
        else
        {
            printf("position %d is not a middle position",pos);
        }
}
void delete_at_beg()
{
        node *temp;
        if(start==NULL)
        {
            printf("\n no. nodes are exist..");
            return;
        }
        else
        {
            temp=start;
            start=start->next;
            free(temp);
            printf("\n node deleted");
        }
```

```
}
void delete_at_last()
{
        node *temp,*prev;
        if(start==NULL)
        {
                printf("\n empty list..");
                return;
        }
        else
        {
                temp=start;
                prev=start;
                while(temp->next!=NULL)
                {
                        prev=temp;
                        temp=temp->next;
                }
                prev->next=NULL;
                free(temp);
                printf("\n node existed");
        }
}
```

```
void delete_at_mid()
{
        int ctr=0,pos,nodectr;
        node *temp,*prev;
        if(start==NULL)
        {
                printf("\n empty list..");
                return;
        }
        else
        {
                printf("\n enter position of node to delete..");
                scanf("%d",&pos);
                nodectr=countnode(start);
                if(pos>nodectr)
                {
                        printf("\n this node does not exist");
                }
                if(pos>1&&pos<nodectr)
                {
                        temp=prev=start;
                        ctr=1;
                        while(ctr<pos)
```

```
                {
                        prev=temp;
                        temp=temp->next;
                        ctr++;
                }
                prev->next=temp->next;
                free(temp);
                printf("\n node deleted..");
        }
        else
        {
                printf("\n invalid position..");

        }
    }
}
void traverse()
{
    node *temp;
    temp=start;
    printf("\n the contents of list(left to right):\n");
    if(start==NULL)
    {
```

```
            printf("\n empty list");
            return;
        }
        else
            while(temp!=NULL)
            {
                printf("%d-->",temp->data);
                temp=temp->next;
            }
        printf("x");
}
void rev_traverse(node*start)
{
        if(start==NULL)
        {
            return;
        }
        else
        {
            rev_traverse(start->next);
            printf("%d-->",start->data);
        }
}
```

```
int countnode(node *st)
{
        if(st==NULL)
                return 0;
        else
                return(1+countnode(st->next));
}
void main(void)
{
        int ch,n;

        while(1)
        {
                ch=menu();
                switch(ch)
                {
                case 1: if(start==NULL)
                {
                        printf("\n no. of nodes you want to create");
                        scanf("%d",&n);
                        createlist(n);
                        printf("\n list created..");
                }
```

```
else
{
        printf("\n list is already created..");
}
break;
case 2: insert_at_beg();
break;
case 3: insert_at_end();
break;
case 4: insert_at_mid();
break;
case 5: delete_at_beg();
break;
case 6: delete_at_last();
break;
case 7: delete_at_mid();
break;
case 8: traverse();
break;
case 9: printf("\n the contents of list(right to left):\n");
rev_traverse(start);
break;
case 10: printf("\n no. of nodes:%d",countnode(start));
```

```
                break;
                case 11: exit(0);
                }

        }
}
```

Input & Output:

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:1

no. of nodes you want to create3

enter data:4

enter data:5

enter data:6

list created..

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:8

the contents of list(left to right):

4-->5-->6-->x

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:9

the contents of list(right to left):

6-->5-->4-->

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:2

enter data:3

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:8

the contents of list(left to right):

3-->4-->5-->6-->x

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:10

no. of nodes:4

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:5

node deleted

1.create a list

2.insert a node at beginning

3.insert a node at end

4.insert a node at middle

5.delete a node from beginning

6.delete a node from list

7.delete a node at middle

8.traverse the list(left to right)

9. traverse the list(right to left)

10.count nodes

11.exit

enter your choice:8

the contents of list(left to right):

4-->5-->6-->x

Week-11

1) **Write a C program that uses functions to perform the following operations on Double linked list.**
i) Creation
ii) Insertion
iii) Deletion
iv) Traversal

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   struct node *prev;
   int n;
   struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();

int count = 0;

void main()
{
   int ch;
   clrscr();
```

```
h = NULL;
temp = temp1 = NULL;

printf("\n 1 - Insert at beginning");
printf("\n 2 - Insert at end");
printf("\n 3 - Insert at position i");
printf("\n 4 - Delete at i");
printf("\n 5 - Display from begining");
printf("\n 6 - Display from end");
printf("\n 7 - Search for element");
printf("\n 8 - Sort the list");
printf("\n 9 - Update an element");
printf("\n 10 - Exit");

while (1)
{
   printf("\n Enter choice : ");
   scanf("%d", &ch);
   switch (ch)
   {
   case 1:
      insert1();
      break;
   case 2:
      insert2();
      break;
   case 3:
      insert3();
      break;
   case 4:
      delete();
      break;
   case 5:
      traversebeg();
      break;
   case 6:
      temp2 = h;
      if (temp2 == NULL)
         printf("\n Error : List empty to display ");
```

```
            else
            {
                printf("\n Reverse order of linked list is : ");
                traverseend(temp2->n);
            }
            break;
        case 7:
            search();
            break;
        case 8:
            sort();
            break;
        case 9:
            update();
            break;
        case 10:
            exit(0);
        default:
            printf("\n Wrong choice menu");
        }
    }
}

/* TO create an empty node */
void create()
{
    int data;

    temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node : ");
    scanf("%d", &data);
    temp->n = data;
    count++;
}

/*  TO insert at beginning */
void insert1()
```

```
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}

/* To insert at end */
void insert2()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp1->next = temp;
        temp->prev = temp1;
        temp1 = temp;
    }
}

/* To insert at any position */
void insert3()
{
    int pos, i = 2;
```

```
    printf("\n Enter position to be inserted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Position out of range to insert");
        return;
    }
    if ((h == NULL) && (pos != 1))
    {
        printf("\n Empty list cannot insert other than 1st position");
        return;
    }
    if ((h == NULL) && (pos == 1))
    {
        create();
        h = temp;
        temp1 = h;
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        create();
        temp->prev = temp2;
        temp->next = temp2->next;
        temp2->next->prev = temp;
        temp2->next = temp;
    }
}

/* To delete an element */
void delete()
{
```

```
int i = 1, pos;

printf("\n Enter position to be deleted : ");
scanf("%d", &pos);
temp2 = h;

if ((pos < 1) || (pos >= count + 1))
{
   printf("\n Error : Position out of range to delete");
   return;
}
if (h == NULL)
{
   printf("\n Error : Empty list no elements to delete");
   return;
}
else
{
   while (i < pos)
   {
      temp2 = temp2->next;
      i++;
   }
   if (i == 1)
   {
      if (temp2->next == NULL)
      {
         printf("Node deleted from list");
         free(temp2);
         temp2 = h = NULL;
         return;
      }
   }
   if (temp2->next == NULL)
   {
      temp2->prev->next = NULL;
      free(temp2);
      printf("Node deleted from list");
      return;
```

```
        }
        temp2->next->prev = temp2->prev;
        if (i != 1)
            temp2->prev->next = temp2->next;    /* Might not need this statement if i
== 1 check */
        if (i == 1)
            h = temp2->next;
        printf("\n Node deleted");
        free(temp2);
    }
    count--;
}

/* Traverse from beginning */
void traversebeg()
{
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("List empty to display \n");
        return;
    }
    printf("\n Linked list elements from begining : ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}

/* To traverse from end recursively */
void traverseend(int i)
{
    if (temp2 != NULL)
    {
        i = temp2->n;
```

```
        temp2 = temp2->next;
        traverseend(i);
        printf(" %d ", i);
    }
}

/* To search for an element in the list */
void search()
{
    int data, count = 0;
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("\n Error : List empty to search for data");
        return;
    }
    printf("\n Enter value to search : ");
    scanf("%d", &data);
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
            printf("\n Data found in %d position",count + 1);
            return;
        }
        else
            temp2 = temp2->next;
            count++;
    }
    printf("\n Error : %d not found in list", data);
}

/* To update a node value in the list */
void update()
{
    int data, data1;

    printf("\n Enter node data to be updated : ");
```

```
    scanf("%d", &data);
    printf("\n Enter new data : ");
    scanf("%d", &data1);
    temp2 = h;
    if (temp2 == NULL)
    {
        printf("\n Error : List empty no node to update");
        return;
    }
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {

            temp2->n = data1;
            traversebeg();
            return;
        }
        else
            temp2 = temp2->next;
    }

    printf("\n Error : %d not found in list to update", data);
}

/* To sort the linked list */
void sort()
{
    int i, j, x;

    temp2 = h;
    temp4 = h;

    if (temp2 == NULL)
    {
        printf("\n List empty to sort");
        return;
    }
```

```
    for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
    {
        for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
        {
            if (temp2->n > temp4->n)
            {
                x = temp2->n;
                temp2->n = temp4->n;
                temp4->n = x;
            }
        }
    }
    traversebeg();
}
```

Week-12

12) Write a C program that uses functions to perform the following operations on circular linked list.

i) Creation
ii) Insertion
iii) Deletion
iv) Traversal

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
```

```
    int data;
    struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
    int choice =0;
    while(choice != 7)
    {
        printf("\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");
        printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from
Beginning\n4.Delete from last\n5.Search for an
element\n6.Show\n7.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            beginsert();
            break;
            case 2:
```

```
            lastinsert();
            break;
            case 3:
            begin_delete();
            break;
            case 4:
            last_delete();
            break;
            case 5:
            search();
            break;
            case 6:
            display();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
```

```
        printf("\nEnter the node data?");
        scanf("%d",&item);
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }

}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
```

```
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp -> next != head)
            {
                temp = temp -> next;
            }
            temp -> next = ptr;
            ptr -> next = head;
        }

        printf("\nnode inserted\n");
    }

}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
```

```
        free(head);
        printf("\nnode deleted\n");
    }

    else
    {   ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nnode deleted\n");

    }
}
void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");

    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
```

```
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");

    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
        printf("item found at location %d",i+1);
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
```

```
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        }
        if(flag != 0)
        {
            printf("Item not found\n");
        }
    }

}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");
```

```
        while(ptr -> next != head)
        {

            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }

}
```

OUTPUT:

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
1

Enter the node data?10

node inserted

*********Main Menu*********

Choose one option from the following list ...

```
==============================================

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
2

Enter Data?20

node inserted

*********Main Menu*********

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
2

Enter Data?30

node inserted

*********Main Menu*********

Choose one option from the following list ...
```

```
==============================================

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
3

node deleted

*********Main Menu*********

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
4

node deleted

*********Main Menu*********

Choose one option from the following list ...

==============================================

1.Insert in begining
```

2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
5

Enter item which you want to search?
20
item found at location 1
*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
6

printing values ...
20

*********Main Menu*********

Choose one option from the following list ...

==========

1.Insert in begining
2.Insert at last
3.Delete from Beginning

4.Delete from last
5.Search for an element
6.Show
7.Exit

Enter your choice?
7

Week-13

13) Write a C program that uses functions to perform the following:

i) Creating a Binary Tree of integers

ii) Traversing the above binary tree in preorder, inorder and postorder.

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct node {
   int data;

   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data) {
   struct node *tempNode = (struct node*) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   //if tree is empty
   if(root == NULL) {
      root = tempNode;
   } else {
      current = root;
      parent = NULL;

      while(1) {
         parent = current;

         //go to left of the tree
         if(data < parent->data) {
            current = current->leftChild;

            //insert to the left
            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }  //go to right of the tree
         else {
```

```
            current = current->rightChild;

            //insert to the right
            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
}

struct node* search(int data) {
   struct node *current = root;
   printf("Visiting elements: ");

   while(current->data != data) {
      if(current != NULL)
         printf("%d ",current->data);

      //go to left tree
      if(current->data > data) {
         current = current->leftChild;
      }
      //else go to right tree
      else {
         current = current->rightChild;
      }

      //not found
      if(current == NULL) {
         return NULL;
      }
   }

   return current;
}

void pre_order_traversal(struct node* root) {
```

```
   if(root != NULL) {
      printf("%d ",root->data);
      pre_order_traversal(root->leftChild);
      pre_order_traversal(root->rightChild);
   }
}

void inorder_traversal(struct node* root) {
   if(root != NULL) {
      inorder_traversal(root->leftChild);
      printf("%d ",root->data);
      inorder_traversal(root->rightChild);
   }
}

void post_order_traversal(struct node* root) {
   if(root != NULL) {
      post_order_traversal(root->leftChild);
      post_order_traversal(root->rightChild);
      printf("%d ", root->data);
   }
}

int main() {
   int i;
   int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

   for(i = 0; i < 7; i++)
      insert(array[i]);

   i = 31;
   struct node * temp = search(i);

   if(temp != NULL) {
      printf("[%d] Element found.", temp->data);
      printf("\n");
   }else {
      printf("[ x ] Element not found (%d).\n", i);
   }
```

```
  i = 15;
  temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }

  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);

  return 0;
}
```

OUTPUT:

```
Visiting elements: 27 35 [31] Element found.
Visiting elements: 27 14 19 [ x ] Element not found (15).

Preorder traversal: 27 14 10 19 35 31 42
Inorder traversal: 10 14 19 27 31 35 42
Post order traversal: 10 19 14 31 42 35 27
```

Week-14

14) Write C programs that use both recursive and non-recursive functions to perform the following searching operations for a key value in a given list of integers:

i) Linear search
ii) Binary search

A) Program to find Linear Search

```
#include <stdio.h>

#include <stdlib.h>

int main(void) {

        int a[10],i,n,s=1,f;

        printf("\n enter the size of array ::");

        scanf("%d",&n);

        printf("\n enter the  array elements:");
```

```
for(i=0;i<n;i++)
        scanf("%d",&a[i]);
printf("\n enter the searching element::");
scanf("%d",&f);
for(i=0;i<n;i++)
{
        if(f==a[i])
        {
                printf("\n %d is position searching element %d is found ",i+1,s);
        s=2;
        break;
        }
}
        if(s==1)
                printf("\n searching element is not found in arrays");
 }
```

Output:

```
 enter the size of array ::3
 enter the  array elements:6
9
7
 enter the searching element::9
```

2 is position searching element 1 is found

enter the size of array ::2

enter the array elements:6

3

enter the searching element::6

1 is position searching element 1 is found

B) Program to find Binary Search

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
        int a[10],i,n,m,L,h,s=1,f=1;
```

```
        printf("\n enter the array size::");
        scanf("%d",&n);
printf("\n enter the sorted array eLements in ascending order ::");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        printf("\n enter the searching eLement::");
        scanf("%d",&f);
        L=0;
        h=n-1;
        whiLe(L<=h)
        {
                m=(L+h)/2;
                if(f<a[m])
                        h=m-1;
                eLse if(f>a[m])
                                L=m+1;
                eLse
                {
                        printf("\n eLement %d is position %d",f,m+1);
                        s=2;
                        break;
                }
        }
```

```
        if(s==1)
                printf("\n searching eLement is not found");
}
```

Output:

1. enter the array size::3

enter the sorted array Elements in ascending order ::3

6

7

enter the searching Element::6

Element 6 is position 2

2. enter the array size::2

enter the sorted array Elements in ascending order ::5

9

enter the searching Element::9

Element 9 is position 2

Week-15

15) Write a C program that implements the following sorting methods to sort a given list of integers in ascending order

i) Bubble sort
ii) Selection sort
iii) Insertion sort

A) Program to find bubble Sort

```
#include <stdio.h>
#include <stdlib.h>
void bubblesort(int x[],int n)
{
        int i,j,t;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n-1;j++)
                {
                        if(x[j]>x[j+1])
                        {
                                t=x[j];
                                x[j]=x[j+1];
                                x[j+1]=t;
```

```
                    }
                }
            }
}
int main(void)
{
        int i,n,x[25];
        printf("\n enter the no.of elements:");
        scanf("%d",&n);
        printf("\n enter the elements:");
        for(i=0;i<n;i++)
                scanf("%d",&x[i]);
        bubblesort(x,n);
        printf("\n array elements after sorting:");
        for(i=0;i<n;i++)
                printf("%5d",x[i]);
}
```

Output:

enter the no.of elements:3

enter the elements:6

3

1

array elements after sorting: 1 3 6

B) Program to find selection sort

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
        int i,j,t,n,min,a[10];
        printf("\n enter the size of array:");
        scanf("%d",&n);
        printf("\n enter the array elements:");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
        for(i=0;i<n;i++)
```

```
        {
                min=i;
                for(j=i+1;j<n;j++)
                {
                        if(a[min]>a[j])
                        {
                                min=j;
                        }
                        if(i!=min)
                        {
                                t=a[i];
                                a[i]=a[min];
                                a[min]=t;
                        }
                }
        }
        printf("\n sorted order(selection sort) is:");
        for(i=0;i<n;i++)
                printf("%4d",a[i]);
}
```

Output:

enter the size of array:3

enter the array elements:3

6

1

sorted order(selection sort) is: 1 3 6

C) Program to find Insertion sort

```
#include <stdio.h>
#include<conio.h>
void main()
{
        int n,a[10],i,pos,temp;
```

```
        clrscr();
        printf("\nEnter number of elements you want to sort:");
        scanf("%d", &n);
        printf("\nEnter %d integers:", n);
        for (i=0;i<n;i++)
                scanf("%d", &a[i]);
        for (i=1;i<n;i++)
        {
                temp=a[i];
                pos = i;
                while ( pos>0 && a[pos-1]>temp)
                {
                        a[pos]=a[pos-1];
                        pos--;
                }
                a[pos]=temp;
        }
        printf("\nSorted list in ascending order:");
        for (i=0; i<n;i++)
                printf("%3d", a[i]);
        getch();
}
```

Output:

Enter number of elements you want to sort: 3

Enter 3 integers: 5 1 9

Sorted list in ascending order: 1 5 9

Report on: Navigating Android - A Practical Exploration

1.Introduction:

In today's digitally driven world, proficiency in using mobile operating systems like Android is essential. This report documents our hands-on exploration of Android's navigation mechanisms. Through practical experimentation, we aimed to gain a comprehensive understanding of the various methods available to users, from gesture controls to traditional button navigation, and to analyze their effectiveness and user-friendliness. This study provides insights into how we, as students, navigated the complex landscape of Android's user interface, and the conclusions we drew from our experiences**.**

2.Methodology:

The "Methodology" section of the Android navigation report is crucial because it establishes the rigor and validity of the study. Here's a breakdown of its key aspects and why they're important:

Key Aspects of a Strong Methodology:

- **Clear Objectives:**
  - It should explicitly state the goals of the exploration. In this case, understanding and comparing Android navigation methods.
- **Device Selection and Justification:**
  - It should detail the devices used (models and Android versions) and explain why they were chosen. This ensures diverse representation and allows for comparative analysis.
- **Standardized Procedures:**
  - It should outline the specific steps taken to test each navigation method (gestures, buttons, app drawer, settings, in-app, accessibility). This ensures consistency and replicability.
- **Data Collection Methods:**
  - It should describe how data was collected (e.g., observation, screenshots, timing tasks). This provides transparency and allows for verification.
- **Comparative Analysis:**
  - It should explain how different navigation methods and device configurations were compared. This demonstrates a systematic approach to evaluating performance.
- **Accessibility Testing:**
  - It should detail how accessibility features were tested, and how the student recorded the results of those tests.
- **Documentation:**

    - It should mention the use of documentation tools (e.g., screenshots, notes) to ensure accurate recording of observations.

Why a Strong Methodology Is Important:

- **Reliability:**
    - A well-defined methodology ensures that the findings are reliable and not based on subjective impressions.
- **Validity:**
    - It strengthens the validity of the conclusions by demonstrating that the study was conducted in a systematic and unbiased manner.
- **Replicability:**
    - It allows others to replicate the study and verify the findings.
- **Transparency:**
    - It provides transparency into the research process, allowing readers to understand how the conclusions were reached.
- **Credibility:**
    - It enhances the credibility of the report by demonstrating a rigorous and scientific approach.

In the context of the Android navigation report, a strong methodology ensures that:

- The comparisons between gesture and button navigation are fair and accurate.
- The observations about app drawer and settings organization are based on systematic exploration.
- The evaluation of accessibility features is thorough and objective.
- The differences between manufactures, and android versions are clearly shown.

By providing a detailed and well-structured methodology, the report establishes its credibility and provides valuable insights into the complexities of Android navigation.

Device Selection and Setup:

The "Device Selection and Setup" section within the methodology is a critical foundation for the Android navigation report. Here's a deeper look at its importance and key considerations:

Importance of Device Selection and Setup:

- **Representation of the Android Ecosystem:**
    - Android runs on a vast array of devices from different manufacturers, each with its own customizations. Selecting a variety of devices ensures that the study reflects the diversity of the Android experience.
    - It avoids a biased view based on a single device or manufacturer.
- **Android Version Diversity:**
    - Android versions introduce significant changes to navigation and user interface. Including devices with different Android versions allows for the observation of these changes.
    - It helps to identify trends and potential compatibility issues.
- **Control and Consistency:**

    - Resetting devices to factory settings provides a controlled environment for testing. This eliminates any pre-existing customizations or app installations that could influence the results.
    - It ensures that all tests are conducted under similar conditions.
- **Comparative Analysis:**
    - Using multiple devices allows for a direct comparison of navigation methods across different hardware and software configurations.
    - This is very important for a study that is meant to show the differences between manufacturers.

Key Considerations for Device Selection and Setup:

- **Device Variety:**
    - Include devices from different manufacturers (e.g., Google, Samsung, Motorola, etc.).
    - Include devices with different form factors (e.g., smartphones, tablets).
    - Include devices with varying hardware specifications (e.g., different processors, screen sizes).
- **Android Version Range:**
    - Select devices running a range of Android versions, including both recent and older versions.
    - This helps to capture the evolution of Android navigation.
- **Clear Documentation:**
    - Precisely document the make, model, and Android version of each device used.
    - This ensures that the study can be replicated and that the results can be properly interpreted.
- **Consistent Setup:**
    - Ensure that all devices are set up in a consistent manner (e.g., same language, same network connection).
    - This minimizes the impact of extraneous variables on the results.
- **Navigation Configuration:**
    - Explicitly state whether gesture navigation or three-button navigation (or both) were enabled on each device.
    - This is essential for understanding the context of the navigation tests.
- **Network Stability:**
    - Ensuring a stable network connection, is important, especially when testing applications that rely on internet access.

Gesture Navigation Testing:

The "Gesture Navigation Testing" section is a crucial part of the Android navigation report, as it delves into the modern and increasingly prevalent method of interacting with Android devices. Here's a breakdown of its importance and key aspects:

Importance of Gesture Navigation Testing:

- **Modern User Interface:**
  - Gesture navigation represents a significant shift in how users interact with mobile devices. Testing it provides insights into the effectiveness of this modern UI paradigm.
- **User Experience Evaluation:**
  - It allows for the evaluation of the user experience associated with gesture navigation, including its intuitiveness, efficiency, and comfort.
- **Identification of Strengths and Weaknesses:**
  - It helps to identify the strengths and weaknesses of gesture navigation, such as its space efficiency versus its potential for misinterpretation.
- **Compatibility Assessment:**
  - It enables the assessment of gesture navigation compatibility with different apps and launchers, highlighting potential conflicts or inconsistencies.
- **Manufacturer Variations:**
  - It allows the student to see the differences between how the different manufacturers have implemented the gesture navigation.

Key Aspects of Gesture Navigation Testing:

- **Specific Gesture Testing:**
  - The report should clearly specify which gestures were tested (e.g., swiping back, swiping home, swiping recent apps).
  - It should describe the precise execution of each gesture.
- **Testing Environment:**
  - The report should outline the testing environment, including the apps used and the tasks performed.
- **Observation and Documentation:**
  - The report should describe how observations were recorded, such as noting the ease of execution, responsiveness, and any errors or misinterpretations.
  - Use of screenshots and videos to record test results.
- **Subjective and Objective Data:**
  - The report should include both subjective data (e.g., user opinions on ease of use) and objective data (e.g., number of errors, time taken to complete tasks).
- **App Compatibility Testing:**

    - The report should detail the testing of gesture navigation within a variety of apps, including both system apps and third-party apps.
    - The student should note any conflicts with app UI elements.
- **Sensitivity and Responsiveness:**
    - The report should note the sensitivity of the gesture controls, and the responsiveness of the Android OS.

Potential Challenges and Considerations:

- **Learning Curve:**
    - Acknowledge the learning curve associated with gesture navigation and account for it in the testing process.
- **Subjectivity:**
    - Recognize that user experience with gestures can be subjective and strive for objective measurements where possible.
- **Device Variations:**
    - Account for variations in gesture implementation across different devices and android versions.

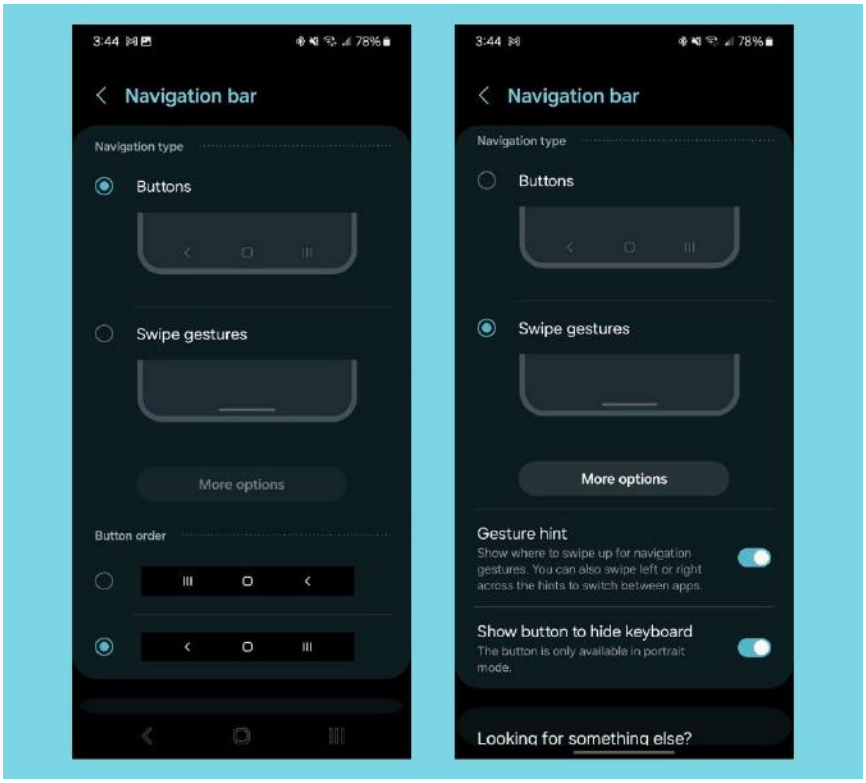
Three-Button Navigation Testing:

The "Three-Button Navigation Testing" section of the Android navigation report is designed to assess the effectiveness and user experience of the traditional navigation method. Here's a breakdown of its key aspects and significance:

Key Aspects of Three-Button Navigation Testing:

- **Standardized Task Performance:**
    - The methodology should specify the tasks performed using the three-button navigation. Examples include:
        - Switching between recently opened apps.
        - Returning to the home screen from various apps.
        - Using the "back" button to navigate within apps and settings menus.
- **Time Measurement (Optional but Valuable):**
    - If possible, timing the completion of these tasks can provide objective data on the efficiency of three-button navigation.
    - This allows for a direct comparison with gesture navigation.
- **Usability Observations:**
    - Documenting observations about the ease of use, responsiveness, and reliability of the buttons.
    - Noting any instances of accidental button presses or difficulties in reaching the buttons.
- **Consistency Checks:**
    - Assessing the consistency of the button functions across different apps and settings menus.
    - Ensuring that the "back" button behaves predictably.
- **Comparative Analysis:**
    - Explicitly comparing the three button navigation to the gesture navigation.
    - Noting the pros and cons of each system.

Significance of Three-Button Navigation Testing:



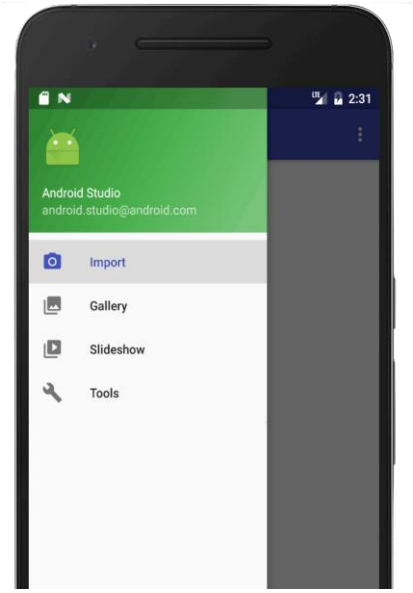
- **Baseline for Comparison:**
  - Three-button navigation serves as a familiar and consistent baseline for comparing the newer gesture navigation.
  - It allows for a clear evaluation of the advantages and disadvantages of each method.
- **User Familiarity:**
  - Many users are accustomed to three-button navigation, making it important to assess its continued relevance.
  - It helps to understand the transition challenges for users switching to gesture navigation.
- **Reliability Assessment:**
  - Three-button navigation is known for its reliability. Testing its performance ensures that it continues to function as expected.
  - This is especially important when testing on various devices.
- **Accessibility Considerations:**
  - For some users with accessibility needs, physical buttons may be easier to use than gestures.
  - The test can help to see what accessibility problems may arise.
- **Understanding User Preferences:**
  - The gathered data can show if users prefer the old style of navigation, or if they prefer the new style.

App Drawer and Settings Navigation:

The "App Drawer and Settings Navigation" section of the Android navigation report delves into two fundamental aspects of user interaction: accessing applications and managing system configurations. Here's a breakdown of its importance and key considerations:

Importance of App Drawer and Settings Navigation:

- **Application Access:**
  - The app drawer is the primary gateway to all installed applications. Its organization and search functionality directly impact user efficiency.
  - Understanding how users interact with the app drawer is crucial for evaluating the overall usability of the Android interface.
- **System Configuration:**
  - The settings menu allows users to customize their devices and manage system functions. Its structure and organization play a vital role in user experience.
  - Analyzing settings navigation reveals how easily users can find and modify system settings.
- **Manufacturer Customization:**
  - Android manufacturers often customize the app drawer and settings menu. Examining these customizations highlights variations in user interface design.
  - This section allows for the report to point out the differences between stock android, and other manufacturer versions of android.



- **User Efficiency:**
  - This section allows for the report to investigate how efficiently a user can find applications, and settings.

Key Considerations for App Drawer and Settings Navigation:

- **App Drawer Functionality:**

    - **Search Efficiency:** Evaluate the accuracy and speed of the app drawer's search function.
    - **Organization:** Assess the ease of organizing apps into folders or other organizational structures.
    - **Access Methods:** Document different methods of accessing the app drawer (e.g., swipe up, app icon).
- **Settings Menu Structure:**
    - **Organization Logic:** Analyze the logical organization of the settings menu (e.g., categories, subcategories).
    - **Search Effectiveness:** Evaluate the accuracy and relevance of the settings search function.
    - **Setting Accessibility:** Assess the ease of finding specific settings, particularly those related to display, sound, network, and accessibility.
- **Manufacturer Variations:**
    - **UI Customization:** Document any significant differences in the app drawer and settings menu UI across different manufacturers.
    - **Feature Availability:** Note any variations in the availability of specific features or settings.
- **User Experience:**
    - **Intuitiveness:** Evaluate the overall intuitiveness of the app drawer and settings navigation.
    - **Efficiency:** Assess the speed and efficiency of completing common tasks, such as finding an app or changing a setting.
- **Documentation:**
    - Use screenshots and detailed notes to document the app drawer and settings menu structure.
    - Record the steps taken to find specific settings or perform specific tasks.

In-App Navigation Analysis:

The "In-App Navigation Analysis" section is vital for understanding how navigation principles translate into real-world user experiences within Android applications. Here's a breakdown of its significance and key aspects:

Significance of In-App Navigation Analysis:

- **Real-World Application:**
    - It goes beyond system-level navigation to examine how users interact with specific apps, which is where most user activity occurs.
    - It provides insights into the practical application of navigation principles.
- **User Experience Evaluation:**
    - It allows for the assessment of how effectively apps guide users through their features and content.
    - It reveals whether navigation is intuitive, efficient, and user-friendly.
- **Identification of Navigation Patterns:**
    - It helps to identify common navigation patterns used by app developers, such as bottom navigation bars, side menus, and tabbed interfaces.
    - It allows for the evaluation of the effectiveness of these patterns.
- **Usability Assessment:**

- It contributes to a broader usability assessment of Android apps, highlighting potential areas for improvement.
- It shows how well apps adhere to android design guidelines.

Key Aspects of In-App Navigation Analysis:

- **App Selection:**
  - Choose a diverse range of popular apps that represent different categories (e.g., social media, productivity, entertainment, mapping).
  - Select apps that are widely used and relevant to the target audience.
- **Navigation Pattern Identification:**
  - Identify and document the various navigation patterns used within each app (e.g., bottom navigation bars, side menus, tabbed interfaces, search functions).
  - Note the location, appearance, and functionality of these patterns.
- **Usability Evaluation:**
  - Assess the intuitiveness and ease of use of each navigation pattern.
  - Evaluate the efficiency of navigation in completing common tasks within the app.
  - Note any instances of confusion, frustration, or difficulty.
- **Consistency Analysis:**
  - Examine the consistency of navigation patterns within each app and across different apps.
  - Note any inconsistencies that could lead to user confusion.
- **Search Functionality Assessment:**
  - Evaluate the effectiveness of search functions within apps, including accuracy, relevance, and ease of use.
  - Test the apps back button functionality.
- **Documentation:**
  - Use screenshots and detailed notes to document observations and findings.
  - Organize findings in a clear and structured manner.
- **User flow analysis:**
  - Follow common user flows within the application, and note how easy or difficult those flows were to follow.

Accessibility Feature Exploration:

The "Accessibility Feature Exploration" section is vital for a comprehensive analysis of Android navigation. It goes beyond typical user experience considerations to examine how the OS caters to users with diverse needs. Here's a breakdown of its importance and key aspects:

Importance of Accessibility Feature Exploration:

- **Inclusivity:**
  - It highlights Android's commitment to making technology accessible to everyone, regardless of disability.
  - It demonstrates the practical application of accessibility features in real-world scenarios.
- **User Diversity:**
  - It acknowledges that users have varying abilities and needs.
  - It expands the scope of the study to include a wider range of user experiences.

- **Evaluation of Effectiveness:**
    - It assesses the functionality and usability of accessibility features, identifying their strengths and weaknesses.
    - It provides insights into how these features can be improved.
- **Ethical Considerations:**
    - It underscores the ethical responsibility of technology developers to create inclusive products.
    - It shows that the people conducting the test are thinking of all potential users.

Key Aspects of Accessibility Feature Exploration:

- **Feature Selection:**
    - Focus on features directly related to navigation, such as TalkBack (screen reader), magnification gestures, and button customization.
    - Consider features relevant to different types of disabilities (e.g., visual, motor, cognitive).
- **Systematic Testing:**
    - Develop a structured approach to testing each feature, documenting the steps taken and the results observed.
    - Test the features in various contexts, such as navigating the home screen, using apps, and accessing settings.
- **Usability Evaluation:**
    - Assess the ease of use and effectiveness of each feature.
    - Note any challenges or limitations encountered during testing.
- **User Experience Perspective:**
    - Consider the user experience from the perspective of a person with a disability.
    - Evaluate the clarity of audio feedback (TalkBack), the precision of magnification gestures, and the ease of button customization.
- **Documentation:**
    - Document the functionality and usability of each feature with clear and concise language.
    - Include screenshots or videos to illustrate the features in action.
    - Note any errors, or bugs that were found while testing.
- **Real World Scenarios:**
    - Test accessibility in common app usage. Such as reading an email, watching a video, or using a map application.

Specific Examples of What to Explore:

- **TalkBack:**
    - How accurately does it read screen content?
    - How easy is it to navigate using TalkBack gestures?
    - How well does it integrate with different apps?
- **Magnification Gestures:**
    - How precise are the magnification controls?
    - How easy is it to navigate while zoomed in?
    - How well does it work with different screen elements?
- **Button Customization:**
    - How flexible are the customization options?

    - How easy is it to assign functions to different buttons?
    - How useful are the custom buttons in daily use?

Comparative Analysis:

The "Comparative Analysis" section is the heart of the Android navigation report, as it's where the raw data and observations are transformed into meaningful insights. Here's a breakdown of its importance and key elements:

Importance of Comparative Analysis:

- **Identification of Patterns and Trends:**
    - It allows for the identification of patterns and trends in Android navigation across different devices, Android versions, and manufacturers.
    - This reveals the strengths and weaknesses of different navigation approaches.
- **Evaluation of User Experience:**
    - It enables a direct comparison of the user experience associated with different navigation methods.
    - This helps to determine which methods are more intuitive, efficient, and user-friendly.
- **Highlighting Manufacturer Differences:**
    - It reveals the variations in Android implementation across different manufacturers, particularly in settings organization and gesture customization.
    - This provides valuable insights into the impact of manufacturer customizations on the user experience.
- **Understanding Android Version Evolution:**
    - It demonstrates how Android navigation has evolved over time, highlighting the impact of new features and design changes.
    - This helps to understand the progression of the operating system.
- **Providing Objective Conclusions:**
    - It moves beyond subjective impressions by providing objective comparisons based on collected data.
    - This strengthens the validity of the report's conclusions.

Key Elements of Comparative Analysis:

- **Clear Comparison Criteria:**
    - Establish clear criteria for comparing navigation methods (e.g., efficiency, intuitiveness, accessibility).
    - This ensures that the comparisons are consistent and objective.
- **Side-by-Side Comparisons:**
    - Use side-by-side comparisons of screenshots, videos, or data tables to illustrate differences.
    - This makes it easy for readers to visualize and understand the comparisons.
- **Identification of Strengths and Weaknesses:**
    - Clearly identify the strengths and weaknesses of each navigation method or device configuration.
    - This provides a balanced and comprehensive evaluation.
- **Analysis of Variations:**

    - Analyze the reasons behind variations in navigation implementation across different manufacturers and Android versions.
    - This provides deeper insights into the factors that influence the user experience.
- **Data Summarization:**
    - Summarize the data into relevant tables, graphs, or charts.
- **Logical Organization:**
    - Organize the comparative analysis in a logical and coherent manner, making it easy for readers to follow the comparisons.
    - For example, grouping comparisons by navigation type(gesture vs button), or by manufacture.

Examples of Comparative Analysis in the Report:

Here are some concrete examples of how "Comparative Analysis" would be implemented in the Android navigation report, building on the previous points:

1. Gesture vs. Three-Button Navigation Efficiency:

- **Method:**
    - Create a table listing common navigation tasks (e.g., "Open Gmail," "Return to Home Screen," "Switch to Recent App").
    - Time the completion of each task using both gesture navigation and three-button navigation on the same device.
    - Repeat the process on multiple devices.
- **Presentation:**
    - Present the data in a table showing the average time taken for each task with each navigation method.
    - Include a graph illustrating the time differences.
    - Discuss the results, noting which method was consistently faster and in what scenarios.
- **Analysis:**
    - Explain why one method might be faster (e.g., fewer steps, more direct interaction).
    - Note any variations in efficiency across different devices.

2. Settings Menu Organization Across Manufacturers:

- **Method:**
    - Take screenshots of the settings menus from devices by different manufacturers (e.g., Google Pixel, Samsung Galaxy).
    - Focus on specific settings categories (e.g., "Display," "Network & Internet").
    - Compare the layout, naming conventions, and organization of settings within those categories.
- **Presentation:**
    - Present the screenshots side-by-side, highlighting the differences.
    - Create a table listing specific settings and their location in each manufacturer's menu.
    - Use annotations on the screenshots to point out key differences.
- **Analysis:**
    - Discuss the impact of these differences on user experience.

    - Explain which organization is more logical or user-friendly and why.
    - Note how much easier or harder it was to find specific settings.

3. Gesture Navigation Implementation Across Android Versions:

- **Method:**
    - Test gesture navigation on devices running different Android versions (e.g., Android 10, 11, 12, 13).
    - Focus on specific gestures (e.g., "Back," "Home," "Recent Apps").
    - Observe and document any differences in gesture responsiveness, animation, and functionality.
- **Presentation:**
    - Create a table summarizing the differences in gesture implementation across Android versions.
    - Include short video clips demonstrating the gestures on each device.
    - Describe any noticeable animation changes.
- **Analysis:**
    - Explain how Android's gesture navigation has evolved over time.
    - Note any improvements or regressions in gesture performance.
    - Note if older versions lack any newer gesture features.

4. Accessibility Features Comparison:

- **Method:**
    - Test accessibility features (e.g., TalkBack, magnification) on different devices.
    - Note the quality of the feature, and how well it integrates with the devices navigation.
    - Compare the ease of use of the accessibility feature.
- **Presentation:**
    - Create a table showing the devices, and the usability rating of each feature.
    - Note any device specific issues.
- **Analysis:**
    - Explain if some manufacturers have better accessibility integration.
    - Explain which device had the best implementation of the accessibility feature.

3.Documentation:

"Documentation" in the context of the Android navigation report is crucial for maintaining accuracy, transparency, and replicability. Here's a breakdown of its importance and key aspects:

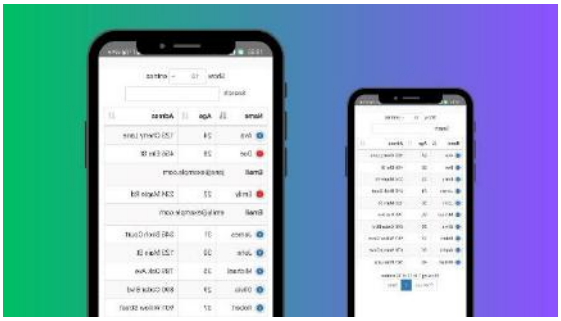
Importance of Documentation:

- **Accuracy and Reliability:**
    - Documentation ensures that observations and findings are recorded accurately, minimizing the risk of errors or misinterpretations.
    - It serves as a reliable record of the research process.
- **Transparency:**
    - It provides transparency into the methodology and findings, allowing readers to understand how the conclusions were reached.

    - It promotes accountability and builds trust in the report.
- **Replicability:**
    - Detailed documentation allows others to replicate the study and verify the findings.
    - This is essential for scientific rigor and validation.
- **Clarity and Organization:**
    - It helps to organize and structure the data, making it easier to analyze and present.
    - It ensures that the report is clear, concise, and easy to follow.
- **Evidence and Support:**
    - Documentation provides evidence to support the claims and conclusions made in the report.
    - It strengthens the credibility of the findings.

Key Aspects of Documentation:

- **Screenshots and Screen Recordings:**
    - Capture visual evidence of navigation interactions, settings configurations, and app interfaces.
    - Use screen recordings to demonstrate gesture movements and navigation flows.
    - Label screenshots and recordings clearly to indicate the device, Android version, and navigation method being demonstrated.
- **Detailed Notes:**
    - Maintain detailed notes on observations, findings, and any challenges encountered during the testing process.
    - Record specific details, such as gesture responsiveness, animation smoothness, and settings menu organization.
    - Note any inconsistencies or unexpected behaviors.
- **Data Tables and Spreadsheets:**
    - Use data tables and spreadsheets to organize and present quantitative data, such as task completion times or user feedback survey results.
    - Ensure that data tables are clearly labeled and include appropriate units of measurement.



- **Version Control:**
    - Document the Android versions and device models used in the study.
    - If the study spans an extended period, document any software updates or changes to the testing environment.

- **Detailed Notes:**
    - Maintain detailed notes on observations, findings, and any challenges encountered during the testing process.

    - Record specific details, such as gesture responsiveness, animation smoothness, and settings menu organization.
    - Note any inconsistencies or unexpected behaviors.
- **Organization and Structure:**
    - Organize documentation in a logical and consistent manner, making it easy to access and review.
    - Use a consistent naming convention for files and folders.
    - Store all documentation in a central, and safe location.
- **User Feedback:**
    - If user feedback is gathered, document the questions asked, and the answers that were given.
    - If possible, record the demographics of the people giving feedback.
- **Timestamping:**
    - Where possible, timestamp data. This will allow for the tracking of changes over time.

Examples of Documentation in the Report:

- Including screenshots of the settings menus from different manufacturers.
- Providing a data table comparing task completion times for gesture and button navigation.
- Including screen recordings demonstrating gesture navigation on different Android versions.
- Providing a list of applications used for in app navigation testing.
- Providing a list of all devices used, and their android versions.

4. Observations and Findings:

Our exploration revealed distinct patterns in Android navigation. Gesture navigation, while initially challenging, offered increased efficiency and screen space, though inconsistencies across devices and apps were noted. Three-button navigation provided reliable familiarity but occupied valuable screen real estate. App drawer search proved highly effective, but settings organization varied significantly by manufacturer, often requiring deep navigation. In-app navigation relied heavily on bottom navigation bars and side menus, with search functionality proving essential for complex apps. Accessibility features like TalkBack were vital, but their integration varied. Device and Android version differences significantly impacted gesture responsiveness, settings layout, and overall navigation consistency, with newer versions generally offering smoother performance and refined controls.

5.Discussion:

Our observations highlight the intricate balance between innovation and user familiarity in Android's navigation design. Gesture navigation, while promising in its efficiency and space optimization, underscores the importance of a smooth learning curve and consistent implementation across the fragmented Android ecosystem. The continued reliance on three-button navigation emphasizes the value of established paradigms, particularly for users seeking predictability. The inconsistencies in settings organization across manufacturers point to a need for greater standardization to enhance user experience. In-app navigation, with its reliance on bottom navigation and side menus, demonstrates the importance of intuitive design for complex applications. Accessibility features, while crucial, reveal the ongoing challenges of ensuring

seamless integration and usability for diverse needs. The variations observed across Android versions and devices highlight the complexities of maintaining a unified user experience within a highly customizable operating system. Ultimately, our findings suggest that Android's navigation design must prioritize user adaptability, consistency, and accessibility, balancing innovation with the need for a reliable and intuitive experience. The future of Android navigation should focus on refining gesture controls, standardizing settings organization, and ensuring seamless integration of accessibility features, while also considering the evolving needs of its diverse user base.

6. Conclusion:

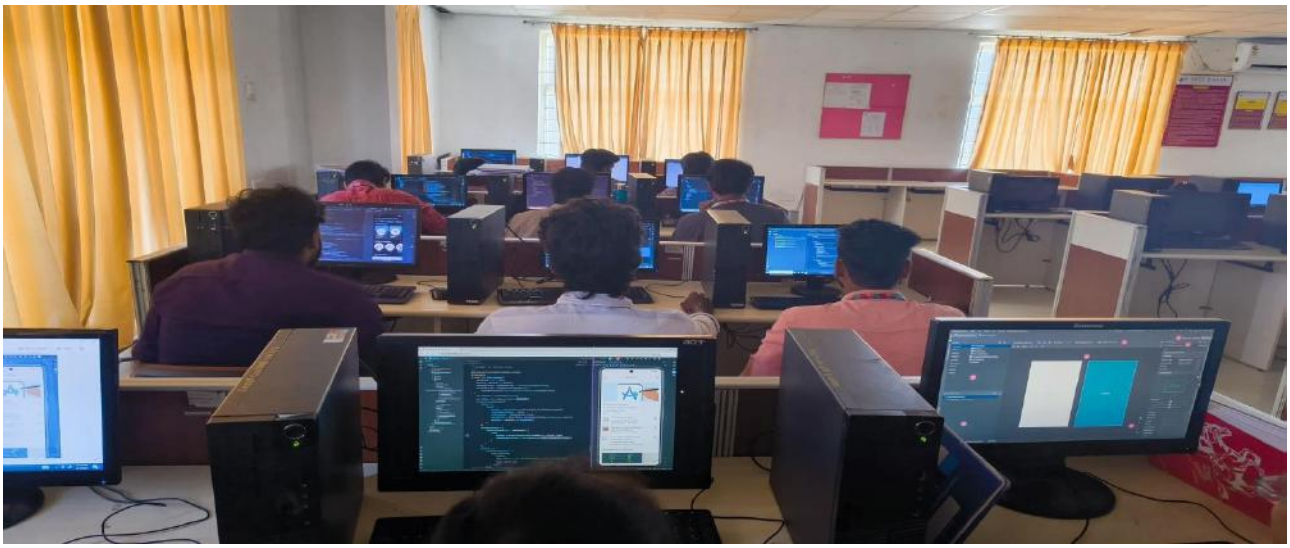
This practical exploration of Android navigation revealed a complex interplay of user experience, device variability, and software design. While gesture navigation offers a modern and efficient approach, its successful implementation hinges on consistent performance and user adaptability. The enduring relevance of three-button navigation underscores the importance of familiarity and reliability. The inconsistencies in settings organization across manufacturers highlight a critical need for standardization within the Android ecosystem. In-app navigation patterns emphasize the necessity of intuitive design principles, while accessibility features demonstrate the ongoing pursuit of inclusivity. Ultimately, the Android navigation experience is a multifaceted one, influenced by user preferences, device capabilities, and software versions. The optimal approach requires a careful balance between innovation and user-friendliness. Future development should prioritize a unified, accessible, and consistently intuitive navigation experience across the diverse Android landscape.

7. Recommendations:

Based on our findings, we recommend the following to improve Android navigation:

- **Standardize Settings Organization:**
  - Google should work with manufacturers to establish clearer guidelines for settings organization, promoting consistency across devices.
  - Consider implementing a more intuitive, searchable, and customizable settings interface.
- **Enhance Gesture Navigation Consistency:**
  - Refine gesture recognition algorithms to improve accuracy and responsiveness across all devices and Android versions.
  - Provide developers with clear guidelines for implementing gesture navigation within their apps to avoid conflicts.
  - Improve gesture sensitivity customization.
- **Improve Accessibility Integration:**
  - Ensure seamless integration of accessibility features (like TalkBack) with all navigation methods.
  - Conduct thorough accessibility testing across all devices and Android versions.
  - Provide more customizable accessibility shortcuts.
- **Optimize In-App Navigation:**
  - Encourage developers to adhere to consistent navigation patterns (e.g., bottom navigation bars for primary functions).
  - Promote the use of robust in-app search functionality.
  - Improve and standardize the back button functionality across applications.

- **Refine Onboarding and Tutorials:**
  - Develop more comprehensive and interactive tutorials for gesture navigation.
  - Provide clear guidance on customizing navigation settings.
  - Offer in app tutorials for complex gesture based applications.
- **Promote User Feedback and Research:**
  - Establish channels for users to provide feedback on navigation experiences.
  - Conduct ongoing research to understand user needs and preferences.
  - Perform comparative studies between Android versions, and other OS navigation.
- **Manufacturer Collaboration:**
  - Promote closer collaboration between google, and device manufacturers, to ensure a unified android experience.



Students working in the Android Lab