String Member Function Prototypes

```
string substr(int pos, int len);
// precondition: 0 <= pos, and pos < length of the string object
// postcondition: returns substring of len characters beginning at position
// pos. Returns as many characters as possible if len is too large, but
// causes error if pos is out of range (>= length of the string object)

int length();  // postcondition: returns the number of characters

int rfind(string s);
// postcondition: rfind is same as find, but searches backwards, returns
the last occurrence // returns string::npos if s does not occur
char at(int pos);
//  you can change or extract one character of a string
// returns the char at specified pos, causes error if pos is out of range
```

Strutils functions

```
int atoi(const string & s);          // returns int equivalent of string s
double atof(const string & s);       // returns double equivalent of string s
string itoa(int n);                  // returns string equivalent of int n
string tostring(int n);              // like itoa, convert int to string
string tostring(double d);           // convert double to string
void ToLower(string & s);            string LowerString(const string & s);
void ToUpper(string & s);            string UpperString(const string & s);
void StripPunc(string & s);          void StripWhite(string & s);
```

```
int find(string s);
// returns first position/index at which substring s begins in, otherwise returns string::npos
int find(string s, int pos);
int rfind(string s, int pos);
// There is another version of find and rfind that takes two parameters
// First parameter is the search string, second parameter is an integer (an pos value)
```

```
class Dice
{
 public:
   Dice(int sides);    // constructor
   int Roll();         // return the random roll
   int NumSides() const;    // number of sides
   int NumRolls() const;    // # times rolled
private:
   int myRollCount;         // # times die rolled
   int mySides;             // # sides on die
};
```

```
RandGen();   // constructor
int RandInt(int max = INT_MAX);
// returns int in [0..max)
int RandInt(int low, int max);
// returns int in [low..max]
double RandReal();
// returns double in [0..1)
double RandReal(double low,
                double max);  // range
                [low..max]
```

Loop Examples

```
int sum = 0;
int i = 1;
while (i <= 10)
{
   sum = sum + i;
   i = i + 1;
}
int sum = 0;
for (int i = 1; i <= 10; i++)
{
    sum = sum + i;
}
do
{
   cout <<"enter number [0..100] ";
   cin >> num;
} while (num < 0 || num > 100 );
```

cin stream – add 10 integers

```
for (count=1; count <= 10; count++)  {
      if (cin >> num)   {
          cout << num << " is valid " <<  endl;
          sum += num;
       }
      else   {
          cin.clear();
          cin >> s;
          cout <<  "entry is invalid“  << endl;
       }
}
```

File Streams

```
ifstream input;                    ofstream out;
string filename =  "test.txt";
input.open(filename.c_str());      // bind input to named file
if  (input.fail() ) {       // if filename is invalid
       cout << "cannot open " << filename << endl;
       return 0;   // stop program
}
while  (  input >> word  ) {
      numWords++;
}
input.clear();       // clear the error flags
input.seekg(0);    // reset the filepos to the beginning of the file
while (  !  input.eof() )     // until the end of the file
{     int num;
     if  ( input >> num )
           cout << num << "\tvalid \n";
     else  {  // clear the error flags and skip the invalid entry
           input.clear();     string s;    input >> s;
           cout << s << "\tinvalid \n";
     }
}
out.open(filename.c_str(), ios::app);  // to append to the end
out << "CS201 test output file " << endl;
for (count=0; count < 10; count++)  {
           out << count +1 << endl;
}  out.close();                        // output file example
// read file line by line                // read file one char at a time
string s;  int num, total=0;             char ch;
while (  getline(input, s)  )            while (  input.get(ch)  )
{       numLines++;                      {         numChars++;
       istringstream ssLine(s);                   if ( '\n' == ch)
       ssLine >> name >> lname;                        numLines++;
       while ( ssLine >> num )                    else if ( '\t' == ch)
             total + num;                              numTabs++;
}                                        }
```

Builtin Array

```
const int MAX_SIZE = 100;
int list[MAX_SIZE];
int k;
list[0] = list[1] = 1;
for (k=2; k < MAX_SIZE, k++)
{
      list[k] = list[k-1]+list[k-2];
}
```

Matrix

```
vector<vector<int>> mat(3, vector<int>(5));
for (int j=0; j < mat[0].size(); j++)   {
     int sum = 0;
     for (int k=0; k < mat.size(); k++)  {
           sum += mat[k][j];
     }
     cout << "sum of column " << j << " is "
          << sum << endl;
}
```

Struct

```
struct student
{
     unsigned int id;
     string name, lastname;
     double gpa;
}
student stu;     stu.name = "Ali";
cout << stu.gpa;
vector<student> class(11);
class[1].gpa = 3.2;
```

char data type

```
cout << "\"\\\n\"\n\\";          char digitch = '3';
char toupper (char ch)  {        int digitnum = digitch - '0';
      if (ch >= 'a' && ch <= 'z')     // if lowercase
            return ch + ('A' - 'a');  // return its uppercase
      return ch;  // otherwise return parameter unchanged
}
```

Robot Member Function Prototypes

```
enum Direction { east, west, north, south };
enum Color { white, yellow, red, blue, green, purple, pink, orange };
class Robot
{
 public:
     Robot (int x, int y, Direction dir = east, int things = 0);
 // robot constructor - color yellow, direction is east and bag count is 0

     void Move (int distance = 1);   // to move robot, default is 1
     void TurnRight ();              // to turn the robot right
     void SetColor (Color color);    // to change the color of robot
     bool FacingEast();              // to check if robot is facing east
     bool FacingWall();      // to check if robot is facing wall
     bool Blocked();         // to check if robot is blocked by  another robot
     bool PickThing ();      // take an item to the bag from current cell
     bool PutThing ();       // put an item to the current cell from bag
     bool CellEmpty ();      // check if the cell is empty
     bool BagEmpty ();       // check if the bag is empty
private:
          int xPos;              //x coordinate of the location of robot
          int yPos;              //y coordinate of the location of robot
          Direction direction;   //current direction of robot
          Color color;           //current color of robot
          int bag;               //current # of things in the bag of robot
          bool stalled;          //true if the robot is dead
          bool visible;          //true if the robot is visible
};
```

```
void ShowMessage (string message);
void ShowMessage (int message);
void GetInput(string prompt, string & var);
void GetInput(string prompt, int & var);
int GetThingCount(int x1,int y1, int x2, int y2);
int GetCellCount (int x, int y);
void PutThings(int xCor, int yCor, int thingCount);
```

Recursion

```
double Power(double x, int n)
// post: returns x^n
{
   if (n == 0)
       return 1.0;

   return x * Power(x, n-1);
}
```

Member Function Examples

```
int Robot::GetXCoordinate()
{
   return xPos;
}
void Robot::Turn(Direction dir)
{
   if (stalled == false)
   {
       direction = dir;
       theRobotWindow-
>Redraw(this);
   }
```

Binary Search

```
int bsearch(const vector<string> & list,
            const string  & key)
{
   int low = 0;
   int high = list.size()-1;
   int mid;
   while (low <= high)   {
      mid = (low + high)/2;
      if (list[mid] == key)    // found
          return mid;
      else if (list[mid] < key)  //upper
          low = mid + 1;
      else    // key in lower half
          high = mid - 1;
   }
   return -1;  // not in list
}
```

The Class Date

```
class Date
{
 public:
    // constructors
    Date();                  // construct date with default value
    Date(long days);         // construct date from absolute #
    Date(int m,int d,int y);     // construct date with specified values

    int Month()       const;    // return month corresponding to date
    int Day()         const;    // return day corresponding to date
    int Year()        const;    // return year corresponding to date
    int DaysIn()      const;    // return # of days in month
    string DayName()    const;    // "monday", "tuesday", ...  "sunday"
    string MonthName()  const;    // "january","february",... "december"
    long Absolute()     const;    // number of days since 1 A.D. for date
    string ToString()   const;    // returns string for date in ascii
    int DaysRemaining()  const;    // return # of remaining days in month

    Date operator ++(int);       // add one day, postfix operator
    Date operator --(int);       // subtract one day, postfix operator
    Date& operator +=(long dx);   // add dx, e.g., jan 1 + 31 = feb 1
    Date& operator -=(long dx);   // subtract dx, e.g., jan 1 - 1 = dec 31
    void SetYear(int);
 private:
     int myDay;              // day of week, 0-6
     int myMonth;            // month, 0-11
     int myYear;             // year in four digits, e.g., 1899
};
```

Vectors

```
vector<int> randStats(7);          RandGen random;
for(k=0; k < n; k++)          // pick all random numbers
{       num = random.RandInt(7);  // between 0 and 6
        randStats[num] = randStats[num] + 1;
}
vector<double> d(10, 3.14); // 10 doubles, all pi
vector<string> words(10);  // 10 strings, all ""
vector<Date> holidays(6);  // 6 today's dates

void Count (vector<int> & counts);       void Print(const vector<int> & counts);
vector<int> Count (istream & input, int & total);  // return from a function
vector<string> words;  //create empty vector
while (input >> w) {
            words.push_back(w);  //adds the next word to the vector
                        //also increases the capacity if necessary
 }
void collect(const vector<string> & a, vector<string> & matches)
{       int k;  // matches contains all elements of a with first letter 'A'
        for (k=0; k < a.size(); k++)  {
            if (a[k].substr(0,1) == "A")
                matches.push_back(a[k]);
        }
}
```

Recursion

```
int RecursFibonacciFixed(int n)
{    // Fixing recursive Fibonacci
   static vector<int> storage(31,0);
   if   (0 == n || 1 == n)       return 1;
   else if  (storage[n] != 0)   return storage[n];
   else    {
              storage[n] = RecursFibonacciFixed (n-1) +
                           RecursFibonacciFixed (n-2);
              return storage[n];
   }
}
```

Selection Sort

```
void SelectSort(vector<int> & a)
{
   int j, k, temp, minIndex, numElts = a.size();
   for(k=0; k < numElts - 1; k++)
   {   minIndex = k;       // min element index
       for(j=k+1; j < numElts; j++)
       {   if (a[j] < a[minIndex])
           {   minIndex = j;    // new min index
           }
       }
       temp = a[k];  // swap min and k-th
       a[k] = a[minIndex];
       a[minIndex] = temp;
   }
}
```

Insertion Sort

```
void InsertSort(vector<string> & a) {
   int k,loc, numElts = a.size();
   for(k=1; k < numElts; k++)
   {
       string hold = a[k];   // insert this element
       loc = k;              // location for insertion
       // shift elements to make room for hold
       while (0 < loc  && hold < a[loc-1])
       {
           a[loc] = a[loc-1];
           loc--;
       }
       a[loc] = hold;
   }
}
```

SQL Basics Cheat Sheet

LearnSQL.com

SQL

SQL, or *Structured Query Language*, is a language to talk to databases. It allows you to select specific data and to build complex reports. Today, SQL is a universal language of data. It is used in practically all technologies that process data.

SAMPLE DATA

COUNTRY

id	name	population	area
1	France	66600000	640680
2	Germany	80700000	357000
...	...	...	...

CITY

id	name	country_id	population	rating
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...	...	...	...	...

QUERYING SINGLE TABLE

Fetch all columns from the `country` table:

```
SELECT *
FROM country;
```

Fetch `id` and `name` columns from the `city` table:

```
SELECT id, name
FROM city;
```

Fetch city names sorted by the `rating` column in the default ASCending order:

```
SELECT name
FROM city
ORDER BY rating [ASC];
```

Fetch city names sorted by the `rating` column in the DESCending order:

```
SELECT name
FROM city
ORDER BY rating DESC;
```

ALIASES

COLUMNS

```
SELECT name AS city_name
FROM city;
```

TABLES

```
SELECT co.name, ci.name
FROM city AS ci
JOIN country AS co
  ON ci.country_id = co.id;
```

FILTERING THE OUTPUT

COMPARISON OPERATORS

Fetch names of cities that have a rating above 3:

```
SELECT name
FROM city
WHERE rating > 3;
```

Fetch names of cities that are neither Berlin nor Madrid:

```
SELECT name
FROM city
WHERE name != 'Berlin'
  AND name != 'Madrid';
```

TEXT OPERATORS

Fetch names of cities that start with a 'P' or end with an 's':

```
SELECT name
FROM city
WHERE name LIKE 'P%'
   OR name LIKE '%s';
```

Fetch names of cities that start with any letter followed by 'ublin' (like Dublin in Ireland or Lublin in Poland):

```
SELECT name
FROM city
WHERE name LIKE '_ublin';
```

OTHER OPERATORS

Fetch names of cities that have a population between 500K and 5M:

```
SELECT name
FROM city
WHERE population BETWEEN 500000 AND 5000000;
```

Fetch names of cities that don't miss a rating value:

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

Fetch names of cities that are in countries with IDs 1, 4, 7, or 8:

```
SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);
```

QUERYING MULTIPLE TABLES

INNER JOIN

`JOIN` (or explicitly **`INNER JOIN`**) returns rows that have matching values in both tables.

```
SELECT city.name, country.name
FROM city
[INNER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

LEFT JOIN

`LEFT JOIN` returns all rows from the left table with corresponding rows from the right table. If there's no matching row, **`NULL`**s are returned as values from the second table.

```
SELECT city.name, country.name
FROM city
LEFT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

RIGHT JOIN

`RIGHT JOIN` returns all rows from the right table with corresponding rows from the left table. If there's no matching row, **`NULL`**s are returned as values from the left table.

```
SELECT city.name, country.name
FROM city
RIGHT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

FULL JOIN

`FULL JOIN` (or explicitly **`FULL OUTER JOIN`**) returns all rows from both tables – if there's no matching row in the second table, **`NULL`**s are returned.

```
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

CROSS JOIN

`CROSS JOIN` returns all possible combinations of rows from both tables. There are two syntaxes available.

```
SELECT city.name, country.name
FROM city
CROSS JOIN country;

SELECT city.name, country.name
FROM city, country;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

NATURAL JOIN

`NATURAL JOIN` will join tables by all columns with the same name.

```
SELECT city.name, country.name
FROM city
NATURAL JOIN country;
```

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

`NATURAL JOIN` used these columns to match rows:
`city.id, city.name, country.id, country.name`
`NATURAL JOIN` is very rarely used in practice.

Componentes de nível superior

A capacidade de enviar mensagens de texto, enviar e-mail, jogar e muito mais é realizada em aplicativos Android por meio de quatro classes de componentes de nível superior: BroadcastReceiver, ContentProvider, Service e Activity. Todos eles são representados por objetos Java.

Componentes de atividade

As atividades fornecem os componentes mais visíveis do aplicativo. Eles apresentam conteúdo na tela e respondem à interação do usuário. `Activity` componentes são os únicos componentes que apresentam conteúdo interativo ao usuário. An `Activity` representa algo que um aplicativo pode fazer, e um aplicativo geralmente "faz" várias coisas – ou seja, a maioria dos aplicativos fornece mais de um arquivo

Visualizações do Android

No Android, `Views` são elementos atômicos e indivisíveis que se desenham na tela. Eles podem exibir imagens, texto e muito mais. Assim como os elementos periódicos se combinam para formar produtos químicos, nós nos combinamos `Views` para formar componentes de design que servem a um propósito para o usuário.

Grupos de visualização do Andro

No Android, `ViewGroups` organize `Views` (e outros `ViewGroups`) para formar designs significativos. `ViewGroups` são o vínculo especial que combina visualizações para formar componentes de design que atendem a um propósito para o usuário. ViewGroups são Views que podem conter outras Views dentro deles.

Arquivos de layout do Android

No Android, cada layout é representado por um arquivo XML. Esses arquivos de texto simples servem como modelo para a interface que nosso aplicativo apresenta ao usuário.

Conversão de arquivo de layout

Quando as Atividades são iniciadas pela primeira vez, elas leem um arquivo de layout e o convertem em um conjunto de objetos Java correspondentes. Esses objetos herdam do objeto View base e podem desenhar-se na tela.

Arquivos de projeto

Os arquivos de projeto Android pertencem a uma das três categorias principais: configuração, código e recurso. Os arquivos de configuração definem a estrutura do projeto, os arquivos de código fornecem a lógica e os arquivos de recursos são praticamente todo o resto.

Editor de layout

O Android Studio Layout Editor inclui duas guias: `Text` e `Design`. A `Text` guia edita o arquivo XML subjacente diretamente e `Design` fornece uma interface de arrastar e soltar para obter os mesmos resultados.

Atributos de layout obrigatórios

No Android, dois atributos são obrigatórios em cada layout: `layout_width` e `layout_height`, e os valores mais comuns para esses atributos são `match_parent` e `wrap_content`.

Layout de restrição

ConstraintLayout organiza seus filhos em relação a si mesmo e a outros filhos (leftOf, rightOf, abaixo, etc.). É mais complexo que um layout linear ou de quadro, mas é muito mais flexível. Também é muito mais eficiente para UIs complexas, pois oferece uma hierarquia de visualização mais plana, o que significa que o Android tem menos processamento para fazer em tempo de execução. Outra vantagem de usar layouts de restrição é que eles foram projetados especificamente para funcionar com o editor de design do Android Studio. Ao contrário dos layouts lineares e de formulário, nos quais você normalmente faz alterações no XML, você cria layouts de restrição visualmente. Você arrasta e solta componentes da GUI na ferramenta de blueprint do editor de design e fornece instruções sobre como cada visualização deve ser exibida.

Layout Linear

No Android, `LinearLayout` organiza seus filhos em linha reta, horizontal ou verticalmente. Se estiver na vertical, eles serão exibidos em uma única coluna, e se estiver na horizontal, eles serão exibidos em uma única linha.

Layouts lineares e de restrição

`LinearLayout` é "de cima para baixo" e determina a posição final para cada criança, enquanto `ConstraintLayout` é principalmente "de baixo para cima", pois exige que seus filhos forneçam atributos relacionados à posição.

AndroidX

AndroidX é uma biblioteca de suporte projetada para trazer novos recursos para dispositivos antigos. AndroidX é uma biblioteca de compatibilidade que oferece recursos exclusivos e as APIs mais recentes para versões anteriores e atuais do Android. O AndroidX permite que dispositivos mais antigos experimentem as novas APIs e a tecnologia Android.

ConstraintLayout exclusivo para AndroidX

Ao contrário do `LinearLayout` Android, disponível na primeira versão do kit de desenvolvimento do Android, `ConstraintLayout` não faz parte de nenhuma versão do Android anterior ou atual e está disponível apenas através do AndroidX.

Namespaces Android

No Android, *os namespaces* ajudam a evitar conflitos de atributos (duas versões do `src` atributo em um único `ImageView`, por exemplo). Três namespaces comuns são: `android`, `tools` e `app`. O `android` namespace pertence ao Android SDK e representa atributos disponíveis imediatamente. O `app` namespace nos permite fazer referência a atributos definidos por bibliotecas externas (por exemplo, AndroidX) e aqueles definidos pela nossa aplicação. O `tools` namespace nos permite definir atributos usados exclusivamente para desenvolvimento, o dispositivo do usuário nunca vê esses atributos.

Identificador de recurso Android

Para que as Views se relacionem ou se restrinjam a seus irmãos, elas precisam de uma maneira de referenciá-los – os identificadores de recursos satisfazem essa necessidade.

Paleta de Componentes

A paleta de componentes fornece uma seleção de elementos que podemos arrastar e soltar diretamente em nosso layout.

Documentação de design de materiais

Ao clicar com o botão direito em qualquer elemento da paleta de componentes, podemos navegar até sua respectiva documentação ou diretrizes do Material Design.

Diretrizes de design de materiais

O propósito e o comportamento da maioria das visualizações, incluindo `CardView`, são inspirados nas diretrizes do Material Design. O Google introduziu o Material Design em 2014 para ajudar a padronizar a aparência de aplicativos da web e móveis em todo o ecossistema do Google.

Processo de inflação de layout

Durante a criação do projeto, o Android Studio associa o `activity_main.xml` arquivo de layout ao objeto MainActivity. Os dois estão associados não pelo nome, mas pelo `setContentView` método encontrado na classe Activity. Este método complexo aceita um identificador de recurso de arquivo de layout, gera as visualizações projetadas e as apresenta na tela - um processo conhecido como inflação de layout

Objetos de evento de movimento do

Os objetos MotionEvent são criados pelo Android após capturar a entrada do usuário na tela sensível ao toque e traduzi-la em dados operáveis. Todas as visualizações podem responder a cliques, arrastar, deslizar e muito mais processando MotionEvents.