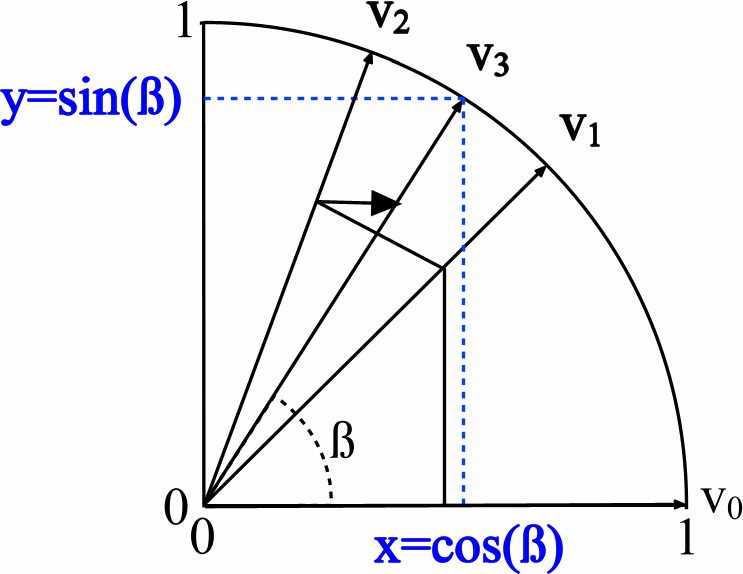
**Exercise 6**: Study about How they creat these below function in standard library:

* Sin (), Cos ().
* Sort (int array[n]).
* Getchar (), putchar ().

1. **Sin (), Cos ().**

CORDIC (for **CO**odinate **R**otation **DI**gital **C**omputer) also know as Volder’s algorithm, is a simple and efficient algorithm to calculate hyperpolic and trigolometric funtion.

To calculate Sin value of a angle β we can use CORDIC, assumes the disired angle is given in radians and represented by a fixed-point fomat. The *y* or *x* coordinate of a point on the unit circle corresponding to the desired angle must be found. Using CORDIC, one would start with the vector {\displaystyle v\_{0}}= .



In the first iteration, this vector is rotated 45° counterclockwise to get the vector . Successive iterations rotate the vector in one or the other direction by size-decreasing steps, until the desired angle has been achieved. Step i size is **actan ()** for **i = 0, 1, 2, …**

More formally, every iteration calculates a rotation, which is performed by multiplying the vector with the rotation matrix :

The rotation matrix is given by:

Using the following two trigonometric identities:

the rotation matrix becomes:

The expression for the rotated vector then becomes:

Where and are the components of . Restricting the angles so that takes on the values , the multiplication with the tangent can be replaced by a division by a power of two, which is efficiently done in digital computer hardware using a bit shift. The expression then becomes:

where

and can have the values of −1 or 1, and is used to determine the direction of the rotation; if the angle is positive then is +1, otherwise it is −1.

After a sufficient number of iterations, the vector's angle will be close to the wanted angle .For most ordinary purposes, 40 iterations (*n* = 40) is sufficient to obtain the correct result to the 10th decimal place.

The only task left is to determine if the rotation should be clockwise or counterclockwise at each iteration (choosing the value of ). This is done by keeping track of how much the angle was rotated at each iteration and subtracting that from the wanted angle; then in order to get closer to the wanted angle , if is positive, the rotation is clockwise, otherwise it is negative and the rotation is counterclockwise.

The values of must also be precomputed and stored. But for small angles, in fixed-point representation, reducing table size.

As can be seen in the illustration above, the sine of the angle is the *y* coordinate of the final vector while the *x* coordinate is the cosine value.

Source: <https://en.wikipedia.org/wiki/CORDIC>

1. **Sort an integer array:**

In standard library, there is a function name qsort().

Following is prototype of qsort():

**void qsort (void\* base, size\_t num, size\_t size,**

**int (\*comparator)(const void\*,const void\*));**

As the name suggests, **the function uses QuickSort algorithm** to sort the given array

The key point about qsort() is comparator function comparator. The comparator function takes two arguments and contains logic to decide their relative order in sorted output. The idea is to provide flexibility so that qsort() can be used for any type (including user defined types) and can be used to obtain any desired order (increasing, decreasing or any other).

The comparator function takes two pointers as arguments (both type-casted to const void\*) and defines the order of the elements by returning (in a stable and transitive manner

int comparator(const void\* p1, const void\* p2);

Return value meaning

<0 The element pointed by p1 goes before the element pointed by p2

=0 The element pointed by p1 is equivalent to the element pointed by p2

>0 The element pointed by p1 goes after the element pointed by p2

Source: <http://www.cplusplus.com/reference/cstdlib/qsort/>