■ INTERFACE DESIGN – Random Number Library

- **Library Characteristics:**

→ Unifying theme • e.g., graphics

→ IF is simple. • simple functions!
("Hide implementation complexity!")

→ IF/LIB must be sufficient/complete. • graphics to support complete drawing functionality

→ Stability of IF/LIB • only extend functionality

⟼

- In stdlib.h: "int rand (void);"

```
/*generates random int. */
/* between 0 and RAND_MAX */
```

- EX:

```
#include <stdio.h>, <stdlib.h>, "genlib.h", ...
#define   NoRandNo   10
main ()
{ int  i, r;
  printf ("RAND_MAX is %d \n", RAND_MAX);
  for (i=0; i < NoRandNo, i++)
  {  r= rand ();
     printf ( "%10 d \n", r );
  }
}
```

→ **SUPPORT MORE RANDOM FUNCTIONS!**

- **EX:** Randomly generate ONE of TWO alternatives!...

  OR

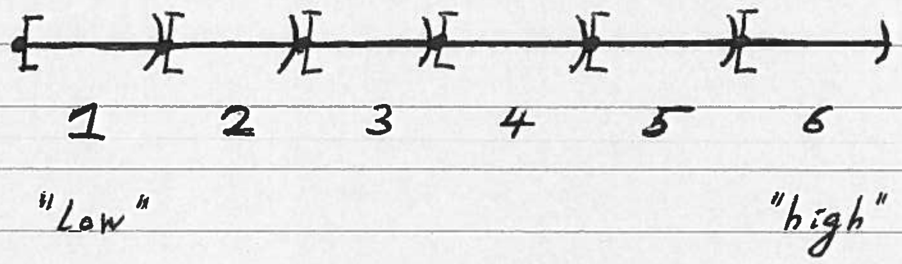
  Rolling a die: SIX possibilities → 1, 2, 3, 4, 5, 6

{ NOTE: RAND_MAX is largest int that can be represented! }

➡ Prototype:

```
int  RandomInt(int low, int high);
                    /* Die: low=1, high=6 */

/* returns random integer from
 * the set {low, ..., high} with
 * equal probability
 */
```

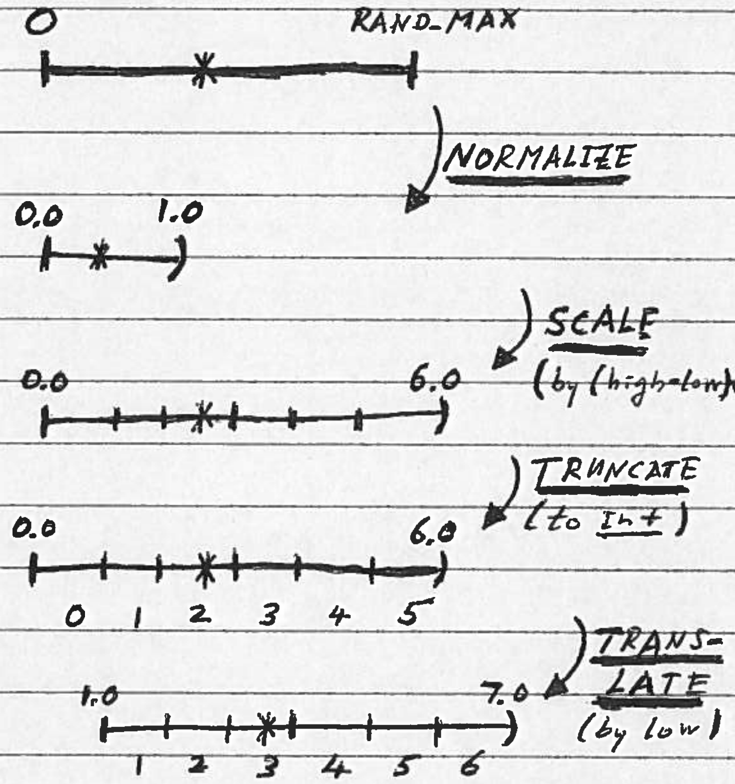
➡ Idea for implementation:



$\boxed{6 = high - low + 1}$

W5

➡ Implemented via the following steps:



```
double  d;
int  D;

d=(double) rand()
   /((double) RAND_MAX+1);
        /* in [0.0,1.0)*
d*= (high-low+1);

D=(int) d;

D+=Low;
```

⬇ C code:

```
int RandomInt(int low, int high)
{ double  d;
  int  D;

   d = (double) rand()/((double) RAND_MAX+1);
   D = (int) (d*(high-low+1));

   return (D+Low);
}
```

Formal Implementation

random.h

```
# ifndef _random_h
# define _random_h
 ...
 /*
  * Function:       RandomInt
  * Input:          2 integers, 'low' and 'high'
  *                 defining set {low,...,high}
  * Output:         a randomly generated int from this set
  * Algorithm:       NOT DESCRIBED HERE!
  * Usage:           how to use function...
  * Last modified:  by whom and when ...
  */
 int RandomInt (int low, int high);
 ...
#endif
```

C code in random.c

random.c

```
/* This library consists of functions
 * needed to simulate random processes.
 */
# include <stdio.h>, <stdlib.h>, "genlib.h",
                "random.h"
...
 /* Function: RandomInt
  * Algorithm : DESCRIBE IN DETAIL!
  */
 int RandomInt (int low, int high)
 {...
 }
```

W5

A CLIENT uses random.h?

```
...
#include "random.h"
#define NoTrial 10
main ()
{ int i;
  for ( i = 0; i < NoTrial; i++)
   { printf("%d \n", RandomInt(1,6));
   }
}
...
```

- **NOTE:** Run 1: 6, 1, 1, 2, 5, 6, 3, 4, 2, 3 } **WHY EQUAL ?**
  Run 2: 6, 1, 1, 2, 5, 6, 3, 4, 2, 3

  → Important for **DE-BUGGING!**

  "**Seed**" → Rand. no. generator → 1st rand. no.

- How to use diffrent seeds?

  → `srand (5) ;  /* seed is 5 */`

  → `srand ((int) time (NULL));`
  ↑ defined in "time.h"

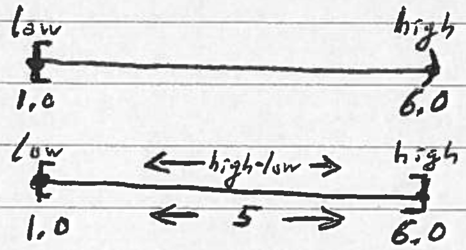
  ↓

```
void RandomizeSeed (void)
{
   srand ((int) time (NULL));
}
```

W5

■ **IF "complete"?** No — Consider also floating-point numbers!

• **EX:**



/* 6.0 not included */

/* 6.0 is included */

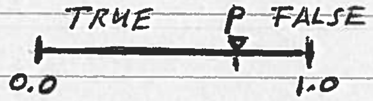
```
double RandomReal (double low, double high)
{
  double d;

    d = ((double) rand() / ((double) RAND_MAX + 1(*)));
    return ( low + d * (high-low));
```

/* (*) + 0 → high included */

/* (*) + 1 → high NOT included */

• **EX:** Function returning TRUE with probability $p \in [0,1]$
(& FALSE with " $(1-p)$)

Idea:

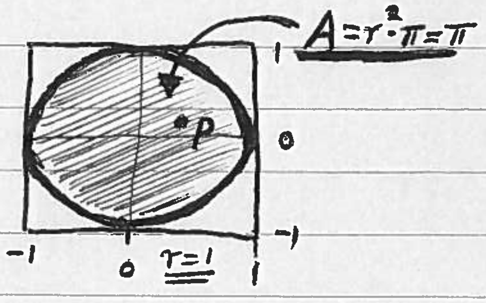


```
bool ReturnTrueWithProb (double p)
{

    return ( RandomReal (0.0, 1.0) < p );
  }
```

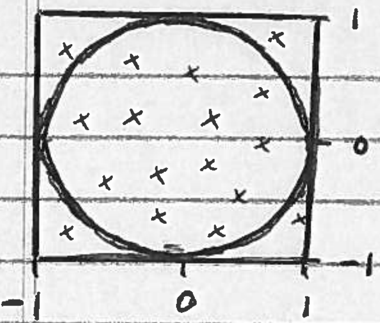
• **EX:** APPROXIMATE 'PI' USING 'MONTE CARLO' METHOD

→ $p = (\bar{x}, \bar{y})$; generate $\bar{x}, \bar{y}$ in $[-1,1]$

→ p inside circle → $\bar{x}^2 + \bar{y}^2 \leq 1.0$



W5



'x': randomly generated points

$\text{Area}_{\square} = 4$

$\text{Area}_{\circ} = \pi$

⇒ $\text{Inside} = \pi$, $\text{Outside} = 4-\pi$

$$\boxed{\frac{\text{Inside}}{\text{Outside}} = \frac{\pi}{4-\pi}} \Rightarrow \underline{\underline{\pi}} = 4\,\frac{\text{Inside}}{\text{Outside}+\text{Inside}}$$

- **NOW:** Inside = No. of 'x' inside circle
  Outside = " " 'x' outside circle

↓

```
...
inside = outside = 0;
for (i=0; i<"infinity"; i++)
{ x = RandReal (-1.0, 1.0);
   y = RandReal (-1.0, 1.0);
   if ( x*x +y*y <= 1.0)
   { inside += 1;
   }
   else
   { outside +=1;
   }
  /* converges to: */
  pi = 4.0*inside/(outside+inside);
  printf ("%f\n", pi);
}
```

⇒ WILL PRINT SEQUENCE OF NUMBERS (SLOWLY) CONVERGING TO VALUE OF PI.