# Survey of Scientific Computing
## (SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

**Session 12**
Continued Fractions,
Chi Squared

1

# Section Goals

- Gain an appreciation for **Continued Fractions** in nature

- Understand the three types of CFs:  1) finite, 2) infinite with repeating *sequence*, 3) infinite with repeating *pattern*

- Write code to **generate** a generalized CF for a real number, and how to **expand** that CF to produce **convergents** of the original number

- Appreciate the hidden underlying simplicity of the **generalized continued fraction** for $\pi$

- Perform a computational mathematical experiment to determine the solutions to **Pell's Equation**

# Seminar Goals

- Gain an appreciation for the **Normal Distribution**

- Investigate if a Normal Distribution can be made from a Uniform Distribution using a **Pachinko game**

- Use **chi-squared statistic** to determine if a random sample conforms to a reasonable Normal Distribution

# Expanding Your Definition of a "Number"
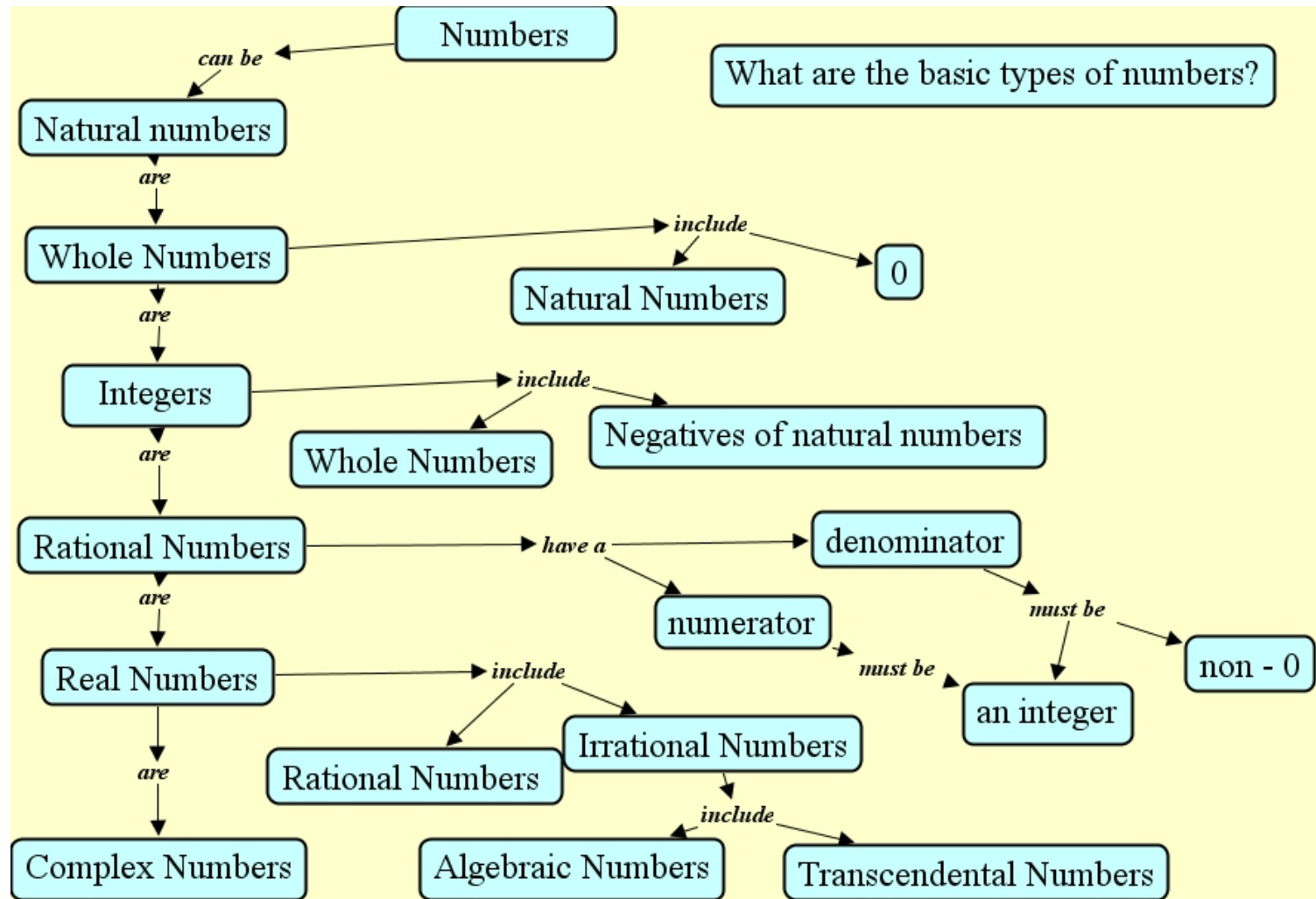
# Continued Fractions

$$x = a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ddots}}}$$

In a <u>simple</u> continued fraction, all $b_n = 1$

$$3.245 = 3 + \cfrac{1}{4 + \cfrac{1}{12 + \cfrac{1}{3 + \cfrac{1}{1}}}}$$

# Continued Fractions

What is the simple CF encoding for **3.245**?

| x | floor(x) | x = x - floor(x) | 1/x |
|---|---|---|---|
| 3.24500000 | 3 | 0.24500000 | 4.08163265 |
| 4.08163265 | 4 | 0.08163265 | 12.25000000 |
| 12.25000000 | 12 | 0.25000000 | 4.00000000 |
| 4.00000000 | 3 | 1.00000000 | 1.00000000 |
| 1.00000000 | 1 | 0.00000000 | |

We stop when this difference is zero

A CF is an *encoding* scheme

**3.245 = [3; 4, 12, 3, 1]**

All **rational** numbers have a CF of **<u>finite</u>** length!

# Continued Fractions

What is the simple CF encoding for **0.825** (= **33/40**)?

| x | floor(x) | x = x - floor(x) | 1/x |
|---:|---:|---:|---:|
| 0.82500000 | 0 | 0.82500000 | 1.21212121 |
| 1.21212121 | 1 | 0.21212121 | 4.71428571 |
| 4.71428571 | 4 | 0.71428571 | 1.40000000 |
| 1.40000000 | 1 | 0.40000000 | 2.50000000 |
| 2.50000000 | 2 | 0.50000000 | 2.00000000 |
| 2.00000000 | 2 | 0.00000000 | |

A CF is an *encoding* scheme

**0.825 = [0; 1, 4, 1, 2, 2]**

All **rational** numbers have a CF of **<u>finite</u>** length!

# Continued Fractions

$h_n = a_n * h_{(n-1)} + h_{(n-2)}$

$k_n = a_n * k_{(n-1)} + k_{(n-2)}$

How do we expand (decode) a simple CF?

**[0; 1, 4, 1, 2, 2] = ??**

$$\Delta = \left( \frac{h_n}{k_n} - x \right)$$

| n | a | h | k | h/k | delta |
|---:|---:|---:|---:|---:|---:|
| -2 | | 0 | 1 | | |
| -1 | | 1 | 0 | | |
| 0 | 0 | 0 | 1 | 0.00000000 | 0.82500000 |
| 1 | 1 | 1 | 1 | 1.00000000 | -0.17500000 |
| 2 | 4 | 4 | 5 | 0.80000000 | 0.02500000 |
| 3 | 1 | 5 | 6 | 0.83333333 | -0.00833333 |
| 4 | 2 | 14 | 17 | 0.82352941 | 0.00147059 |
| 5 | 2 | 33 | 40 | 0.82500000 | 0.00000000 |

Each row gives a better and better approximation **(h / k)** to the original number **x**

**[0; 1, 4, 1, 2, 2] = 0.825** (= **33/40**)

8

# $\sqrt{2}$ to 3,600 digits

1.41421356237309504880168872420969807856967187537694807317667973799073247846210703885038753432764157273501 38
46230912297024924836055850737212644121497099935831413222665927505592755799950501152782060571470109559971 6059
70274534596862014728517418640889198609552329230484308714321450839762603627995251407898687253396546331808 8296
40620615258352395054754750287759961729835575220337531857011354374603408498884716038686999706990048150305 440277
90316454242782306849293691862158057846311159666871301301561856898723723528850926486124949771542183342042 85686
06014682472077143585487415565706967765372022648544701585880162075847492265722600208558446652145839889394 4370
92659180031138824646815708263010059485870400318648034219489727829064104507263688131373985525611732204024 5091
22770022694112757362728049573810896750401836986836845072579936472906076299694138047565482372899718032680 2474
42062926912485905218100445984215059112024944134172853147810580360337107730918286931471017111168391658172 6889
41975871658215212822951848847208969463386289156288276595263514054226765323969461751129160240871551013515 0455
38128756005263146801712740265396947024030051749531886292563138518816347800156936917688185237868405228783 7629
38921430065586956868596459515550164472450983689603688732311438941557665104088391429233811320605243362948 5317
04991577175622854974143899918802176243096520656421182731672625753959471725593463723863226148274262220867 1155
83959992652117625269891754098815934864008345708518147223181420407042650905653233333984364578657967965192 67292
39987536661721598257886026336361782749599421940377775368142621773879919455139723127406689832998989895386 728822
85637869774966251996658352577619893932284534473569479496295216889148549253890475582883452609652409654288 9394
53864662574492755638196441031697983306185201937938494005715633372054806854057586799967012137223947582142 6306
58513221740883238294728761739364746783743196000159218880734785761725221186749042497736692920731109636972 1608
93370866115673458533483329525467585164471075784860246360083444911481858765555428645512331421992631133251 7970
60843655970435285641008791850076036100915946567067688360557174007675690509613671940132493560524018599910 5062
10816359772643138060546701029356997104242510578174953105725593498445112692278034491350663756874776028316 2829
60553242242695753452902883876844642917328277088831808702533985233812274999081237189254072647536785030482 1591
80188616710897286922920119759988070381854333253646021108229927929307287178079988809917674177410898306080 0326
31181642798823117154363869661702999934161614878686018045505553986913115186010386375325004558186044804075 0241
19518430567453368361367459737442398855328517930896037389891517319587413442881784212502191695187559344438 7396
18931454999990610758704909026088351763622474975785885836803745793115733980209998662218694992259591327642 3619
41059210032802614987456659968887406795616739185957288864247346358588686449682238600698335264279905628316 5613
91394255764906206518602164726303336297507569787060660685649816009271870929215313236828135698893709741650 4474
59096053747279652447709409924123871061447054398674364733847745481910087288622214958952959118789214917983 3981
08378827815306556231581036064867587303601450227320882935134138722768417667843690529428698490838455744579 4095
98626074249954916802853077398938296036213353987532050919989360751390644449576845699347127636450716327915 4701
59773354863893942325727754003826027478567417258095141630715959784981800944356037939098559016827215403458 1581
52100493666295344882710729239660232163823826661262683050257278116945103537937156882336593229782319298606 4679
78986409208560955814261436363100461559433255047449397593399912541953230093217530447653396470662761166175 3518
75464620967634558738616488019884849747926404506544489691004079421181692579685756378488149898641685499491 6357
61448404702103398921534237703723335311564594438970365316672194904935188290580630740134686264167247011065 3463
493916407146285

9

# Continued Fractions

What is the simple CF encoding for $\sqrt{2}$ ?

| x | floor(x) | x = x - floor(x) | 1/x |
|---|---|---|---|
| 1.41421356 | 1 | 0.41421356 | 2.41421356 |
| 2.41421356 | 2 | 0.41421356 | 2.41421356 |
| 2.41421356 | 2 | 0.41421356 | 2.41421356 |
| 2.41421356 | 2 | 0.41421356 | 2.41421356 |
| 2.41421356 | 2 | 0.41421356 | 2.41421356 |
| 2.41421356 | 2 | 0.41421356 | 2.41421356 |

$\sqrt{2}$ **= [1; {2}]**

Numbers within {} are repeated

**All irrational numbers yield an infinite CF
with a repeated *sequence* of <u>finite</u> length!**

**There is simple order behind the chaos!**

# Continued Fractions

$$h_n = a_n \cdot h_{(n-1)} + h_{(n-2)}$$

$$k_n = a_n \cdot k_{(n-1)} + k_{(n-2)}$$

$$\Delta = \left( \frac{h_n}{k_n} - x \right)$$

What fraction best approximates $\sqrt{2}$ ?

| n | a | h | k | h/k | delta |
|---:|---:|---:|---:|---:|---:|
| -2 | | 0 | 1 | | |
| -1 | | 1 | 0 | | |
| 0 | 1 | 1 | 1 | 1.00000000 | 0.41421356 |
| 1 | 2 | 3 | 2 | 1.50000000 | -0.08578644 |
| 2 | 2 | 7 | 5 | 1.40000000 | 0.01421356 |
| 3 | 2 | 17 | 12 | 1.41666667 | -0.00245310 |
| 4 | 2 | 41 | 29 | 1.41379310 | 0.00042046 |
| 5 | 2 | 99 | 70 | 1.41428571 | -0.00007215 |
| 6 | 2 | 239 | 169 | 1.41420118 | 0.00001238 |
| 7 | 2 | 577 | 408 | 1.41421569 | -0.00000212 |
| 8 | 2 | 1393 | 985 | 1.41421320 | 0.00000036 |
| 9 | 2 | 3363 | 2378 | 1.41421362 | -0.00000006 |
| 10 | 2 | 8119 | 5741 | 1.41421355 | 0.00000001 |
| 11 | 2 | **19601** | **13860** | 1.41421356 | 0.00000000 |

$$\sqrt{2} \approx \textbf{19,601 / 13,860}$$

# Continued Fractions

What is the simple CF encoding for $\sqrt{113}$ ?

| x | floor(x) | x = x - floor(x) | 1/x |
|---|---|---|---|
| 10.63014581 | 10 | 0.63014581 | 1.58693429 |
| 1.58693429 | 1 | 0.58693429 | 1.70376823 |
| 1.70376823 | 1 | 0.70376823 | 1.42092235 |
| 1.42092235 | 1 | 0.42092235 | 2.37573512 |
| 2.37573512 | 2 | 0.37573512 | 2.66144940 |
| 2.66144940 | 2 | 0.66144940 | 1.51183144 |
| 1.51183144 | 1 | 0.51183144 | 1.95376823 |
| 1.95376823 | 1 | 0.95376823 | 1.04847275 |
| 1.04847275 | 1 | 0.04847275 | 20.63014581 |
| 20.63014581 | 20 | 0.63014581 | 1.58693430 |
| 1.58693430 | 1 | 0.58693430 | 1.70376822 |
| 1.70376822 | 1 | 0.70376822 | 1.42092237 |
| 1.42092237 | 1 | 0.42092237 | 2.37573499 |
| 2.37573499 | 2 | 0.37573499 | 2.66145027 |
| 2.66145027 | 2 | 0.66145027 | 1.51182945 |
| 1.51182945 | 1 | 0.51182945 | 1.95377581 |
| 1.95377581 | 1 | 0.95377581 | 1.04846442 |
| 1.04846442 | 1 | 0.04846442 | 20.63369395 |
| 20.63369395 | 20 | 0.63369395 | 1.57804883 |

Period = 9

$\sqrt{113}$ = [10; {1,1,1,2,2,1,1,1,20}]

# Continued Fractions

$$h_n = a_n * h_{(n-1)} + h_{(n-2)}$$

$$k_n = a_n * k_{(n-1)} + k_{(n-2)}$$

What fraction best approximates $\sqrt{113}$ ?

$$\Delta = \left( \frac{h_n}{k_n} - x \right)$$

| n | a | h | k | h/k | delta |
|---:|---:|---:|---:|---:|---:|
| -2 | | 0 | 1 | | |
| -1 | | 1 | 0 | | |
| 0 | 10 | 10 | 1 | 10.00000000 | 0.63014581 |
| 1 | 1 | 11 | 1 | 11.00000000 | -0.36985419 |
| 2 | 1 | 21 | 2 | 10.50000000 | 0.13014581 |
| 3 | 1 | 32 | 3 | 10.66666667 | -0.03652085 |
| 4 | 2 | 85 | 8 | 10.62500000 | 0.00514581 |
| 5 | 2 | 202 | 19 | 10.63157895 | -0.00143313 |
| 6 | 1 | 287 | 27 | 10.62962963 | 0.00051618 |
| 7 | 1 | 489 | 46 | 10.63043478 | -0.00028897 |
| 8 | 1 | 776 | 73 | 10.63013699 | 0.00000883 |
| 9 | 20 | 16009 | 1506 | 10.63014608 | -0.00000027 |
| 10 | 1 | 16785 | 1579 | 10.63014566 | 0.00000015 |
| 11 | 1 | 32794 | 3085 | 10.63014587 | -0.00000005 |
| 12 | 1 | 49579 | 4664 | 10.63014580 | 0.00000002 |
| 13 | 2 | **131952** | **12413** | 10.63014581 | 0.00000000 |

$\sqrt{113}$ ≈ **131,952 / 12,413**

# *e* to 3,600 digits

2.7182818284590452353602874713526624977572470936999595749669676277240766303535475945713821785251664274274663
9193200305992181741359662904357290033429526059563073813232862793449076323382988075319525101901157383418793070
2154089149934884167509244761460668082264800168477411853742345442437107539077744992069551702761838606261331384
5830007520449338265602976067371132007093287091274437470472306969772093101416928368190255151086574637721112523
8978442505695369677078544996996794686445490598793163688923009877931277361782154249992295763514822082069895193
6680331825288693984964651058209392398294887933203625094431173012381970684161403970198376793206832823764648042
9531180232878250981945581530175671736133206981125099618188159304169035159888851934580727386673858942287922849
9892086805825749279610484198444363463244968487560233624827041978623209002160990235304369941849146314093431738
1436405462531520961836908887070167683964243781405927145635490613031072085103837505101157477041718986106873969
6552126715468895703503540212340784981933432106817012100562788023519303322474501585390473041995777709350366041
6997329725088687696640355570716226844716256079882651787134195124665201030592123667719432527867539855894489696
9709640975459185695638023637016211204774272283648961342251644507818244235294863637214174023889344124796357437
0263755294448337998016125492278509257782562092622648326277933386566481627725164019105900491644998289315056604
7258027786318641551956532442586982946959308019152987211725563474546396447910145904090586298496791287406870504
8958586717479854667757573205681288459205413340539220001137863009455606881667400169842055804033637953764520304
0243225661352783695117788386387443966253224985065499588623428189970773327617178392803494650143455889707194258
6398772754710962953741521115136835062752602326484728703920764310059584116612054529703023647254929666938111513
7322753645098888903136020572481765851180630364428123149655070475102544650117272115551948668508003685322818315
2196003735625279449515828418829478761085263981395599006737648292244375287184624578036192981971399147564488262
6039033814418232625150974827987779964373089970388867782271383605772978824125611907176639465070633045279546618
5509666618566470971134447401607046261568071748187784437143698821855967095910259686200235371858874856965220005
0311734392073211390803293634479727355955277349071783793421637012050054513263835440001863239914907054797780566
9785335580489669062951194324730995876552368128590413832411607226029983305353708761389396391779574540161372236
1878936526053815584158718692553386061647798340254351284396129460352913325942794904337299085731580290958631382
6832914771163963370924003168945863606064584592512699465572483918656420975268508230754425459937691704197778008
5362730941710163434907696423722294352366125572508814779223151974778060569672538017180776360346245927877846585
0656050780844211529697521890874019660906651803516501792504619501366585436632712549639908549144200014574760819
3022120660243300964127048943903971771951806990869986066365832322787093765022601492910115171776359446020232493
0028040186772391028809786660565118326004368850881715723866984224220102495055188169480322100251542649463981287
3677658927688163598312477886520141174110913601164995076629077943464600585194199856016264790761532103872755712
6992518275687989302761761146162549356495903798045838182323368612016243736569846703785853305275833337939907521
6606923805336988795651372855938834998947074161815501253970646481719467083481972144888987906765037959036696724
9499254527903372963616265897603949857674139735944102374432970935547798262961459144293645142861715587339746791
8975712119561873857836447584484235555810500256114923915188930994634284139360803830916628188115037152849670597
4162562823609216807515017772538740256425347087908913729172282861151591568372524163077225440633787593105982676
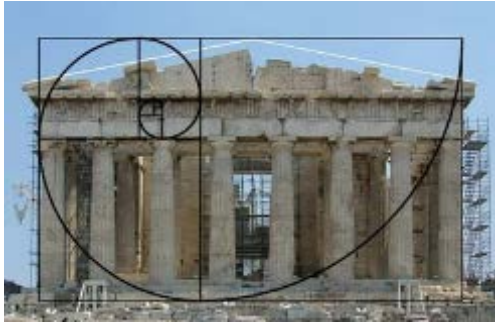0944203261924285317018781772960235413060672136046000389661093647095141417185777014180606443636815

# Continued Fraction for *e*

| x | floor(x) | x = x - floor(x) | 1/x |
|---:|---:|---:|---:|
| 2.71828183 | 2 | 0.71828183 | 1.39221119 |
| 1.39221119 | 1 | 0.39221119 | 2.54964678 |
| 2.54964678 | 2 | 0.54964678 | 1.81935024 |
| 1.81935024 | 1 | 0.81935024 | 1.22047929 |
| 1.22047929 | 1 | 0.22047929 | 4.53557348 |
| 4.53557348 | 4 | 0.53557348 | 1.86715744 |
| 1.86715744 | 1 | 0.86715744 | 1.15319313 |
| 1.15319313 | 1 | 0.15319313 | 6.52770793 |
| 6.52770793 | 6 | 0.52770793 | 1.89498763 |
| | 1 | 0.89498763 | 1.11733388 |
| | 1 | 0.11733388 | 8.52268767 |
| | 8 | 0.52268767 | 1.91318841 |
| | 1 | 0.91318841 | 1.09506427 |
| 1.09506427 | 1 | 0.09506427 | 10.51919947 |
| 10.51919947 | 10 | 0.51919947 | 1.92604201 |
| 1.92604201 | 1 | 0.92604201 | 1.07986461 |
| 1.07986461 | 1 | 0.07986461 | 12.52119027 |
| 12.52119027 | 12 | 0.52119027 | 1.91868508 |
| 1.91868508 | 1 | 0.91868508 | 1.08851229 |

**All transcendental numbers yield an infinite CF with a repeated *pattern* of finite length**

$e = [2; \{1, 2n, 1\}]$ for $n > 0$

$e^2 = [7; 2, \{1, 1, 3n, 12n+6, 3n+2\}]$ for $n > 0$

# The Golden Ratio





$a+b$ is to $a$ as $a$ is to $b$

$$\frac{a+b}{a} = \frac{a}{b} = \varphi$$

$$1 + \frac{b}{a} = \frac{a}{b} = \varphi$$

$$1 + \frac{1}{\varphi} = \varphi$$

$$\varphi + 1 = \varphi^2$$
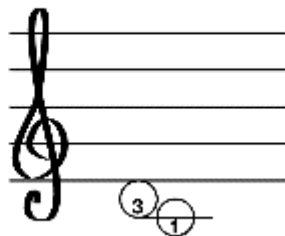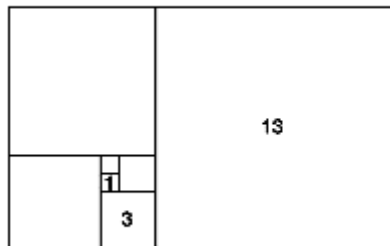
$$\varphi^2 - \varphi - 1 = 0$$
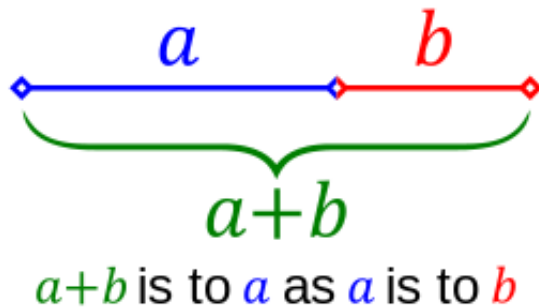
$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \dots$$
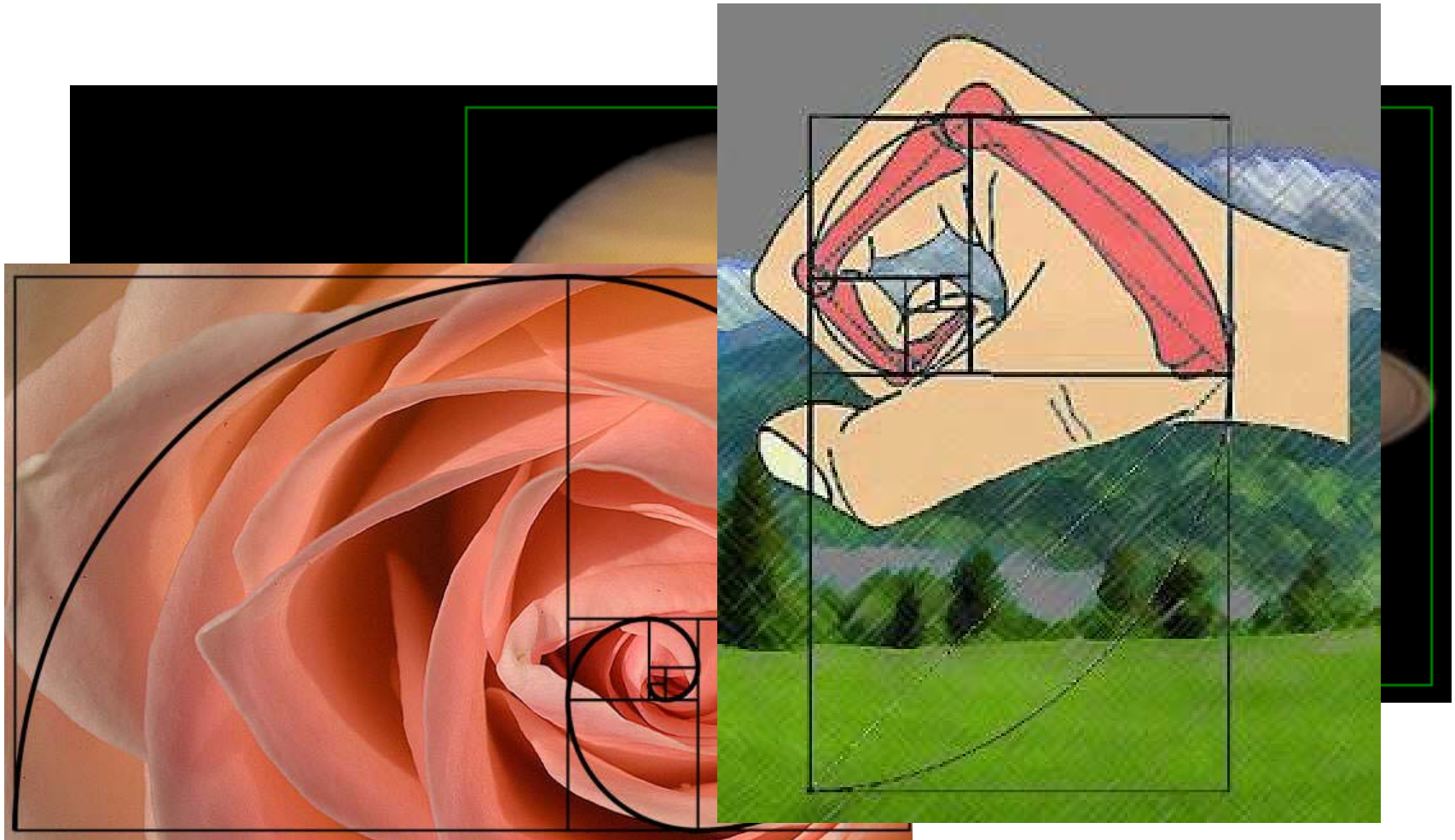
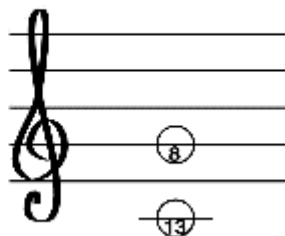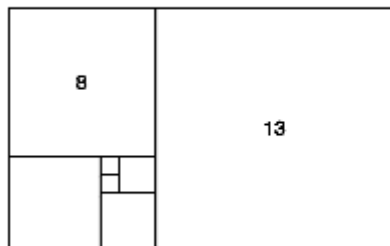$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \dots$$

$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \ldots$$
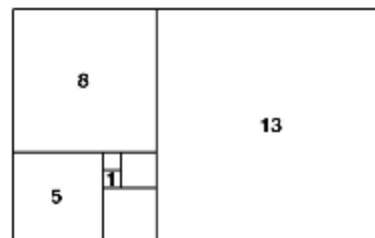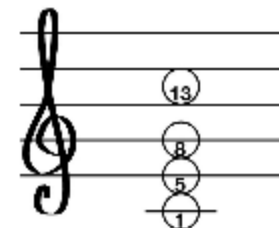


**Whole Step**
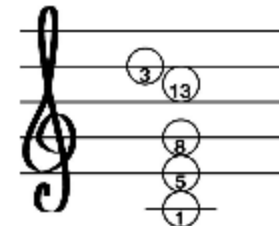
**Perfect Fifth**

**Major Triad**

IDEAL PROPORTIONS  IDEAL CHORD STRUCTURE

**Major 9**

$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887 \ldots$$

The greatest of luthiers, Stradivarius, designed his violins around the golden ratio ($\phi$). His violins are the most valuable and precious instruments in the string-playing world because of their exquisite tonal and harmonic qualities, [2]. The Stradivarius violin in Fig. 2 reveals how precisely his instruments are determined by the golden ratio, [3]:

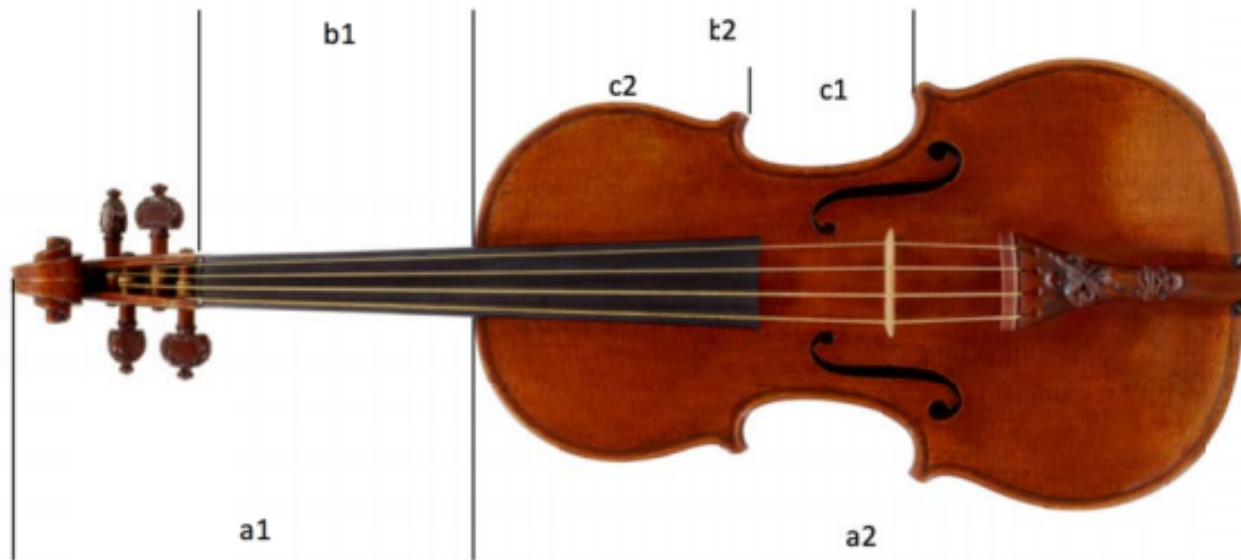$$\frac{a1 + a2}{a2} = \frac{a2}{a1} = \frac{b2}{b1} = \frac{b2}{c2} = \frac{c2}{c1} = \phi$$



Figure 2. Photo of "Lady Blunt" Stradivarius violin (sold for nearly $16M).

# $\varphi$ to 3,600 digits

1.618033988749894848204586834365638117720309179805762862135448622705260462818902449707207204189391137484754088075386891752126633862223536931793180060766726354433389086595939582905638322661319928290267880675208766892501711696207032221043216269548626296313614438149758701220340805887954454749246185695364864449241044320771344947049565846788509874339442212544877066478091588460749988712400765217057517978834166256249407589069704000281210427621771111777805315317141011704666599146697987317613560067087480710131795236894275219484353056783002287856997829778347845878228911097625003026961561700250464338243776486102838312683303724292675263116533924731671112115881863851331620384005222165791286675294654906811317159934323597349498509040947621322298101726107059611645629909816290555208524790352406020172799747175342777592778625619432082750513121815628551222480939471234145170223735805772786160086883829523045926478780178899219902707769038953219681986151437803149974110692608867429622675756052317277752035361393621076738937645560606059216589466759551900400555908950229530942312482355212212415444006470340565734797663972394949946584578873039623090375033993856210242369025138680414577995698122445747178034173126453220416397232134044449487302315417676893752103068737880344170093954409627955898678723209512426893557309704509595684401755519881921802064052905518934947592600734852282101088194644544222318891319294689622002301443770269923007803085261180754519288770502109684249362713592518760777884665836150238913493333122310533923213624319263728910670503399282265263556209029798642472759772565508615487543574826471814145127000602389016207773224499435308899909501680328112194320481964387675863314798571911397815397807476150772211750826945863932045652098969855567814106968372884058746103378105444390943683583581381131168993855576975484149144534150912954070050194775486163075422641729394680367319805861833918328599130396072014455950449779212076124785645916160837059498786006970189409886400764436170933417270919143365013715766011480381430626238051432117348151005590134561011800790506381421527093085880928757034505078081454588199063361298279814117453392731208092897279222132980642946878242748740174505540677875708323731097591511776297844328474790817651809778726841611763250386121129143683437670235037111633072586988325871033632223810980901211019899176841491751233134015273384383723450093478604979294599158220125810459823092552872124137043614910205471855496118087642657651106054588147560443178479858453973128630162544876114852021706440411166076695059775783257039511087823082710647893902111569103927683384538633332156582965977310343603232254574363720412440640888267375843935367959312322134373209957498894699565647360072959998391288103197426312517971414320123112795518947781726914158911779919564812558001845506563295285985910000908621802977563789259991649946428193022293552346674759326951654214021091363018194722707890122087287361707348649998156255472811373479871656952748900814438405327483781378246691744422963491470815700735254570708977267546934382261954686615331209533579238014609273510210119190218360675097308957528957746814229543394385493155339630380729169175846101460995055064803679304147236572039860073550760902317312501613204843583648177048481810991602442523271672190189334596378608787528701739359303013359011237102391712659047026349402830766876743638651327106280323174069317334482343564531850581353108549733350759966778712449058363675413289086240632456395357212524261170278028656043234942837301725574405837278267996031739364013287627701243679831144643694767053127249241047167001382478312865650649343418039004101780533950587724586655755229391582397084177729833728231152

# **Open** Lab 1 – Simple CF Encoding

```cpp
int main()
{
    double x = 3.245;

    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms,"
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```

```cpp
vector<int> EncodeCF(double x)
{
    vector<int> terms;
    while(terms.size() < 20)
    {
        terms.push_back(floor(x));
        x = x - floor(x);
        if (x < 1e-9) break;
        x = 1/x;
    }

    return terms;
}
```

| x | floor(x) | x = x - floor(x) | 1/x |
|---|---|---|---|
| 3.24500000 | 3 | 0.24500000 | 4.08163265 |
| 4.08163265 | 4 | 0.08163265 | 12.25000000 |
| 12.25000000 | 12 | 0.25000000 | 4.00000000 |
| 4.00000000 | 3 | 1.00000000 | 1.00000000 |
| 1.00000000 | 1 | 0.00000000 | |

# View Lab 1 – Simple CF Encoding

```cpp
int main()
{
    double x = 3.245;

    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms, "
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```
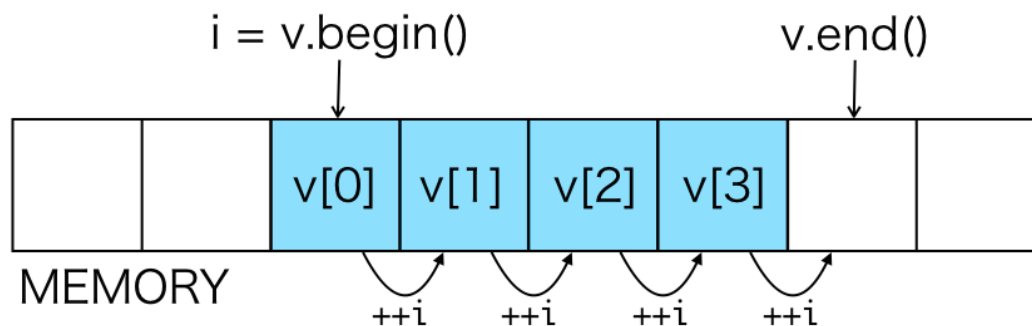
```cpp
void DisplayCF(const vector<int>& terms)
{
    cout << "{";
    auto itr = terms.begin();
    while (true)
    {
        cout << *itr;
        if (++itr == terms.end()) break;
        cout << ", ";
    }
    cout << "}\n";
    return;
}
```

i = v.begin()          v.end()

| | | v[0] | v[1] | v[2] | v[3] | | |
|---|---|---|---|---|---|---|---|

MEMORY     ++i   ++i   ++i   ++i

$v[k] == *(v.begin()+k)$

# **Run** Lab 1 – Simple CF Encoding

```cpp
int main()
{
    double x = 3.245;

    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms, "
        << "the simple continued fraction for\n"
        << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```

```cpp
void DisplayCF(const vector<int>& terms)
{
    cout << "{";
    auto itr = terms.begin();
    while (true)
    {
        cout << *itr;
        if (++itr == terms.end()) break;
        cout << ", ";
    }
    cout << "}\n";
    return;
}
```

Intrinsic data types __cannot__ store most numbers exactly

stdcf-encode

File   Edit   View   Terminal   bs   Help

To 5 terms, the simple continued fraction for
3.24500000000000011 is
{3, 4, 12, 3, 1}

Process returned 0 (0x0)   execution time : 0.031 s
Press ENTER to continue.

| floor(x) |
|---|
| 3 |
| 4 |
| 12 |
| 3 |
| 1 |

# **Edit** Lab 1 –Simple CF Encoding

```cpp
int main()
{
    double x = (1 + sqrt(5)) / 2;

    auto terms = EncodeCF(x);

    cout << "To " << terms.size() << " terms, "
         << "the simple continued fraction for\n"
         << setprecision(18) << x << " is" << endl;

    DisplayCF(terms);

    return 0;
}
```
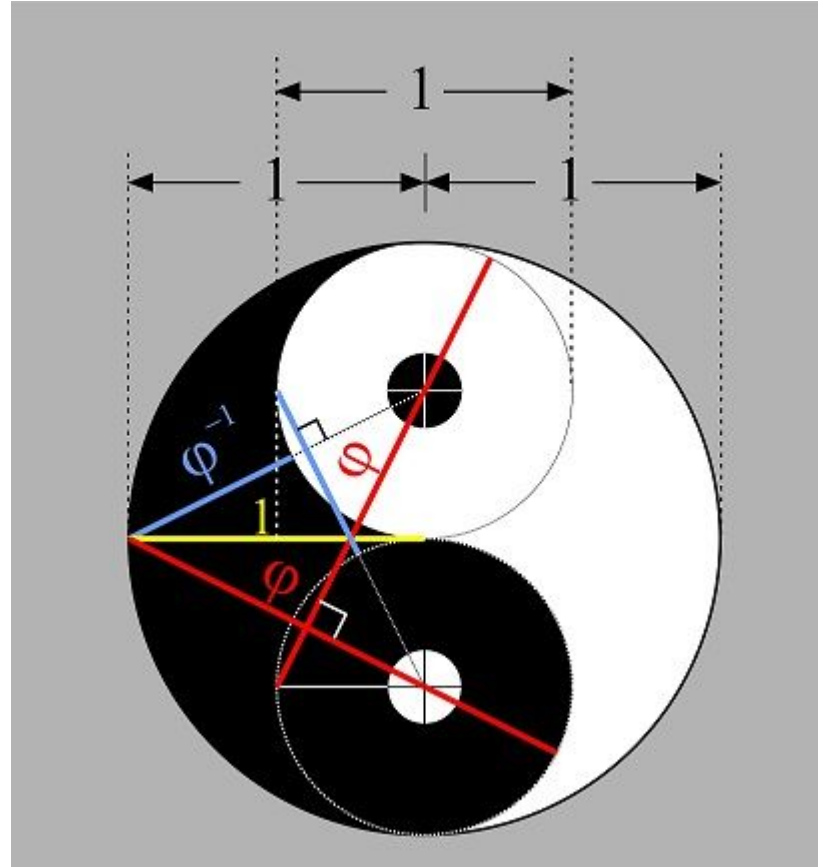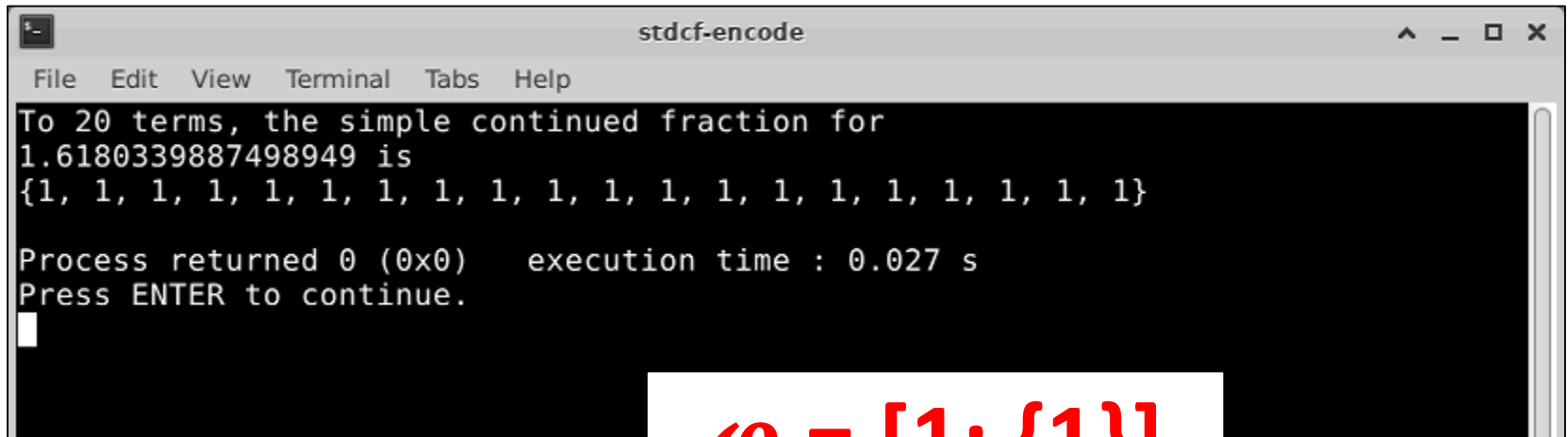
$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.6180339887$$

# **Run** Lab 1 – Simple CF Encoding

- Generate the Simple CF for the golden ratio $\frac{1+\sqrt{5}}{2}$
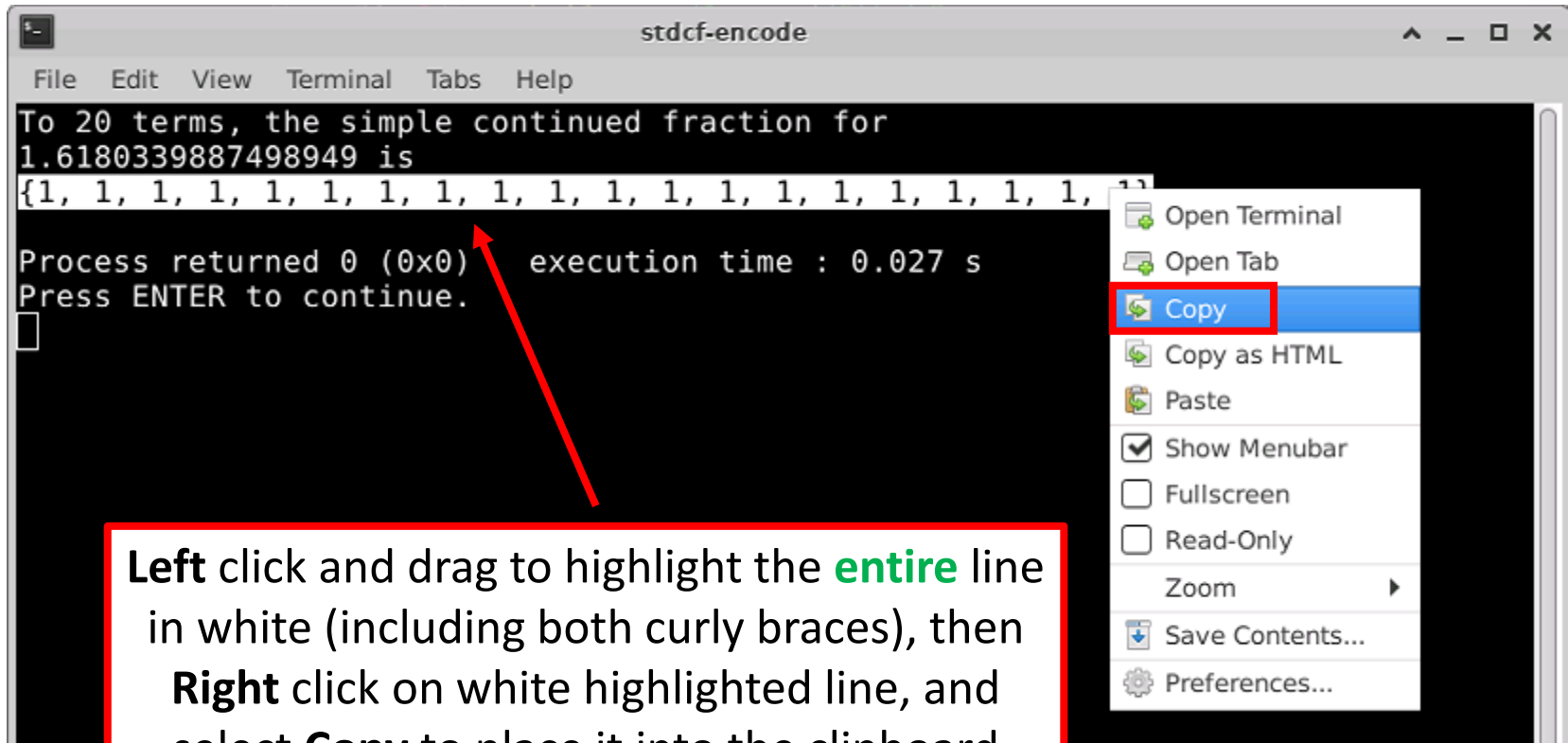
stdcf-encode

File   Edit   View   Terminal   Tabs   Help

```
To 20 terms, the simple continued fraction for
1.6180339887498949 is
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

Process returned 0 (0x0)   execution time : 0.027 s
Press ENTER to continue.
```

**$\varphi$ = [1; {1}]**

This is Mother Nature's true *Unit*
It is the most simple infinite CF possible!

# **Check** Lab 1 – Simple CF Encoding



Left click and drag to highlight the **entire** line in white (including both curly braces), then **Right** click on white highlighted line, and select **Copy** to place it into the clipboard

# C++ Vector Initialization

```
vector<int> cf{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

- Vectors can be defined using the **list initializer** syntax
  - Elements are comma separated between curly braces
  - First item in list goes into index position **0** in the array
  - The vector is dynamically sized to match the number of elements in the initializer list

- Lab 1 emits source code for Lab 2

- Programs <u>can</u> create programs

```cpp
#include "stdafx.h"

using namespace std;

int main()
{
    int maxTerms = 20;

    vector<int> cf{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

    vector<double> h(maxTerms + 2);
    vector<double> k(maxTerms + 2);

    if (cf.size() == 0)
    {
        cout << "Error - Missing cf data!";
        return -1;
    }

    h.at(0) = 0;
    k.at(0) = 1;
    h.at(1) = 1;
    k.at(1) = 0;

    cout << "Using " << maxTerms << " terms, ";
    cout << "the continued fraction expansion is:" << endl;
    cout << setw(5) << "a";
    cout << right << setw(15) << "h";
    cout << right << setw(15) << "k";
    cout << setw(20) << "convergent" << endl;

    for (int n{ 2 }; n < maxTerms + 2; ++n)
    {
        double a = cf.at(n - 2);

        h.at(n) = a * h.at(n - 1) + h.at(n - 2);
        k.at(n) = a * k.at(n - 1) + k.at(n - 2);

        double convergent = h.at(n) / k.at(n);

        cout << setprecision(0) << right
            << setw(5) << a << setw(15) << h[n] << setw(15) << k[n]
            << setprecision(14) << fixed << setw(20) << convergent << endl;
    }

    return 0;
}
```
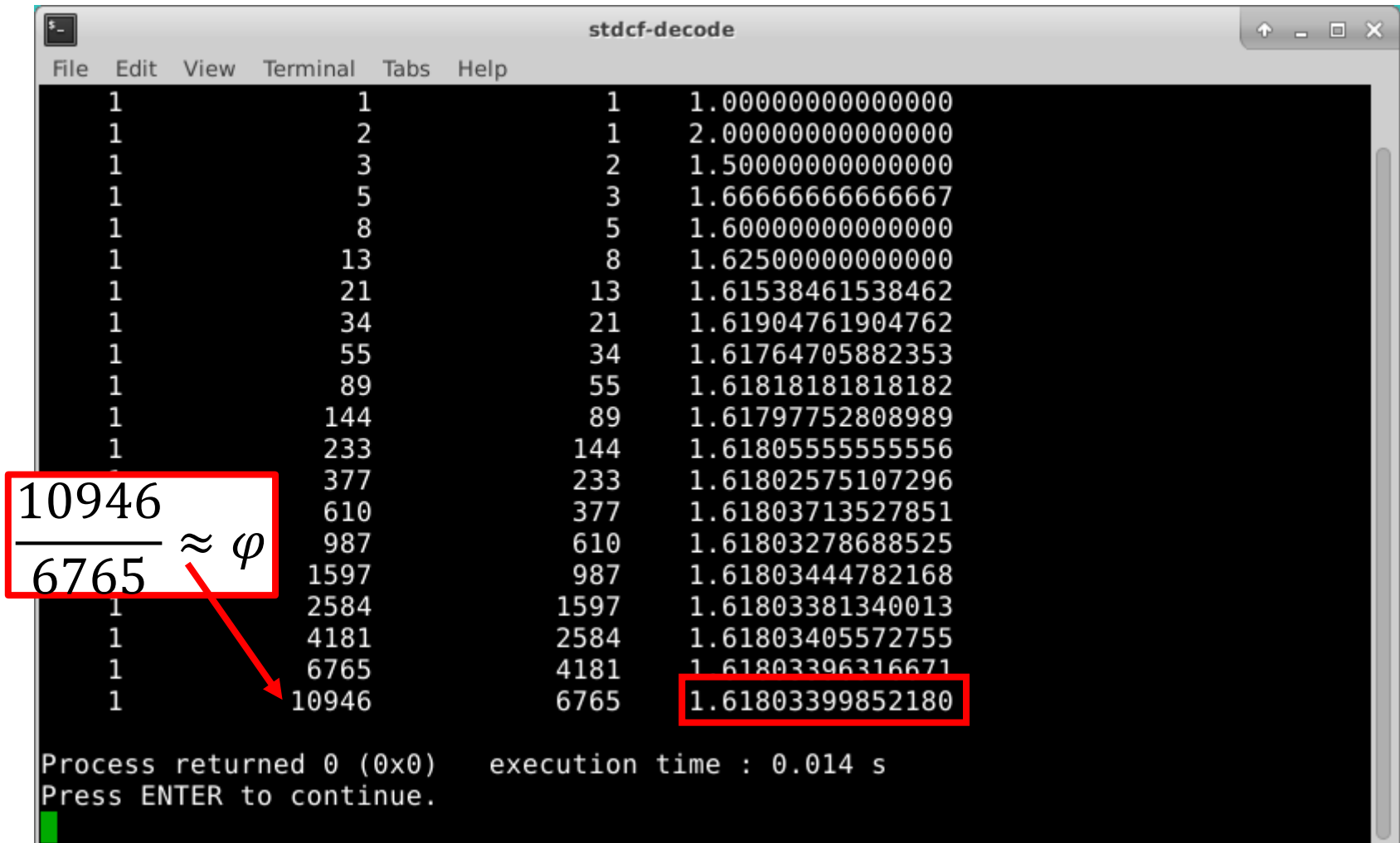
Don't forget the semicolon!

Right click and **paste** in the white line output from Lab 1

$h_n = a_n * h_{(n-1)} + h_{(n-2)}$

$k_n = a_n * k_{(n-1)} + k_{(n-2)}$

# **Run** Lab 2 – Simple CF Decoding



$$\frac{10946}{6765} \approx \varphi$$

# π to 3,600 digits

```
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679821480
8651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566 59
3344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817 48
8152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611 73
8193261179310511854807446237996274956735188575272489122793818301194912983367336244065664308602139494639522 47
3719070217986094370277053921717629317675238467481846766940513200056812714526356082778577134275778960917363 71
7872146844090122495343014654958537105079227968925892354201995611212902196086403441815981362977477130996051 87
0721134999999837297804995105973173281609631859502445945534690830264252230825334468503526193118817101000313 78
3875288658753320838142061717766914730359825349042875546873115956286388235378759375195778185778053217122680 66
1300192787661119590921642019893809525720106548586327886593615338182796823030195203530185296899577362259941 38
9124972177528347913151557485724245415069595082953311686172785588907509838175463746493931925506040092770167 11
3900984882401285836160356370766010471018194295559619894676783744944825537977472684710404753464620804668425 90
6949129331367702898915210475216205696602405803815019351125338243003558764024749647326391419927260426992279 67
8235478163600934172164121992458631503028618297455570674983850549458858692699569092721079750930295532116534 49
8720275596023648066549911988183479775356636980742654252786255181841757467289097777279380008164706001614524 91
9217321721477235014144197356854816136115735255213347574184946843852332390739414333454776241686251898356948 55
6209921922218427255025425688767179049460165346680498862723279178608578438382796797668145410095388378636095 06
8006422512520511739298489608412848862694560424196528502221066118630674427862203919494504712371378696095636 43
7191728746776465757396241389086583264599581339047802759009946576407895126946839835259570982582262052248940 77
2671947826848260147699090264013639443745530506820349625245174939965143142980919065925093722169646151570985 83
8741059788595977297549893016175392846813826868386894277415599185592524595395943104997252468084598727364469 58
4865383673622262609912460805124388439045124413654976278079771569143599770012961608944169486855584840635342 20
7222582848864815845602850601684273945226746767889525213852254995466672782398645659611635488623057745649803 55
9363456817432411251507606947945109659609402522887971089314566913686722874894056010150330861792868092087476 09
1782493858900971490967598526136554978189312978482168299894872265880485756401427047755513237664145152374623 43
6454285844479526586782105114135473573952311342716610213596953623144295248493718711014576540359027993440374 20
0731057853906219838744780847848968332144571386875194350643021845319104848100537061468067491927819119793995 20
6141966342875444064374512371819217999839101591956181467514269123974894090718649423196156794520809514655022 52
3160388193014209376213785595663893778708303906979207734672218256259966150142150306803844773454920260541466 59
2520149744285073251866600213243408819071048633173464965145390579626856100550810665879699816357473638405257 14
5910289706414011097120628043903975951567715770042033786993600723055876317635942187312514712053292819182618 61
2586732157919841484882916447060957527069572209175671167229109816909152801735067127485832228718352093539657 25
1210835791513698820914442100675103346711031412671113699086585163983150197016515116851714376576183515565088 49
0998985998238734552833163550764791853589322618548963213293308985706420467525907091548141654985946163718027 09
8199430992448895757128289059232332609729971208443357326548938239119325974636673058360414281388303203824903 75
8985243744170291327656180937734440307074692112019130203303801976211011004492932151608424448596376698389522 86
84783123552658
```

# Continued Fractions

What is the simple CF encoding for $\pi$ ?

| x | floor(x) | x = x - floor(x) | 1/x |
|---:|---:|---:|---:|
| 3.14159265 | 3 | 0.14159265 | 7.06251331 |
| 7.06251331 | 7 | 0.06251331 | 15.99659441 |
| 15.99659441 | 15 | 0.99659441 | 1.00341723 |
| 1.00341723 | 1 | 0.00341723 | 292.63459088 |
| 292.63459088 | 292 | 0.63459088 | 1.57581844 |
| 1.57581844 | 1 | 0.57581844 | 1.73665853 |
| 1.73665853 | 1 | 0.73665853 | 1.35748105 |
| 1.35748105 | 1 | 0.35748105 | 2.79735107 |
| 2.79735107 | 2 | 0.79735107 | 1.25415271 |
| 1.25415271 | 1 | 0.25415271 | 3.93464232 |
| 3.93464232 | 3 | 0.93464232 | 1.06992802 |
| 1.06992802 | 1 | 0.06992802 | 14.30041960 |
| 14.30041960 | 14 | 0.30041960 | 3.32867763 |
| 3.32867763 | 3 | 0.32867763 | 3.04249485 |
| 3.04249485 | 3 | 0.04249485 | 23.53226532 |
| 23.53226532 | 23 | 0.53226532 | 1.87876228 |
| 1.87876228 | 1 | 0.87876228 | 1.13796418 |
| 1.13796418 | 1 | 0.13796418 | 7.24825805 |
| 7.24825805 | 7 | 0.24825805 | 4.02806683 |

$\pi$ = [3;7,15,1,292,1,1,1,2,1,3,1,14,3,3,23,1,1,7….]
(no repeated *pattern* of finite length ☹!)

# Continued Fractions

| n | a | h | k | h/k | delta |
|---:|---:|---:|---:|---:|---:|
| -2 | | 0 | 1 | | |
| -1 | | 1 | 0 | | |
| 0 | 3 | 3 | 1 | 3.000000000000000 | 0.141592653589793 |
| 1 | 7 | 22 | 7 | 3.142857142857140 | -0.001264489267350 |
| 2 | 15 | 333 | 106 | 3.141509433962260 | 0.000083219627529 |
| 3 | 1 | 355 | 113 | 3.141592920353980 | -0.000000266764189 |
| 4 | 292 | 103993 | 33102 | 3.141592653011900 | 0.000000000577891 |
| 5 | 1 | 104348 | 33215 | 3.141592653921420 | -0.000000000331628 |
| 6 | 1 | 208341 | 66317 | 3.141592653467440 | 0.000000000122356 |
| 7 | 1 | 312689 | 99532 | 3.141592653618940 | -0.000000000029143 |
| 8 | 2 | 833719 | 265381 | 3.141592653581080 | 0.000000000008715 |
| 9 | 1 | 1146408 | 364913 | 3.141592653591400 | -0.000000000001611 |
| 10 | 3 | 4272943 | 1360120 | 3.141592653589390 | 0.000000000000404 |
| 11 | 1 | 5419351 | 1725033 | 3.141592653589820 | -0.000000000000022 |

If measuring the circumference of Earth:

22 / 7 = accurate to between this classroom and Washington, DC

355 / 113 = accurate to between this classroom and the main parking lot

If measuring the distance between Earth & Sun:

355 / 113 = accurate to 4 football fields

104348 / 33215 = accurate to the length of my shoe

# Generalized Continued Fractions

$$x = a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \ddots}}}$$

In a <u>generalized</u> continued fraction,
$a_n$ and $b_n$ can be any expression

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3}}}$$

In a **simple** CF all the
numerators must be 1

# Generalized Continued Fractions

What is a **generalized** CF encoding for $\pi$ ?

**Euler**

$$\pi = 3 + \cfrac{1^2}{6 + \cfrac{3^2}{6 + \cfrac{5^2}{6 + \cfrac{7^2}{6 + \ddots}}}}$$

$a_0 \quad b_0 \quad a_n \quad b_n$

$\pi = [3; 1, \{6 | (2n+1)^2\}]$

All the mysterious and unpredictable digits of PI
come from this simple generalized CF !!

# Generalized Continued Fractions

What is a **generalized** CF expansion for $\pi$ ?

| n | a | b | h | k | h/k | delta |
|---|---|---|---|---|---|---|
| -2 | | | 0 | 1 | | |
| -1 | 0 | 1 | 1 | 0 | | |
| 0 | 3 | 1 | 3 | 1 | 3.00000000 | 0.14159265 |
| 1 | 6 | 9 | 19 | 6 | 3.16666667 | -0.02507401 |
| 2 | 6 | 25 | 141 | 45 | 3.13333333 | 0.00825932 |
| 3 | 6 | 49 | 1321 | 420 | 3.14523810 | -0.00364544 |
| 4 | 6 | 81 | 14835 | 4725 | 3.13968254 | 0.00191011 |
| 5 | 6 | 121 | 196011 | 62370 | 3.14271284 | -0.00112019 |
| 6 | 6 | 169 | 2971101 | 945945 | 3.14088 | |
| 7 | 6 | 225 | 50952465 | 16216200 | 3.14207 | |
| 8 | 6 | 289 | 974212515 | 310134825 | 3.14125 | |
| 9 | 6 | 361 | 20570537475 | 6547290750 | 3.14183962 | -0.0002469 |
| 10 | 6 | 441 | 475113942765 | 151242416325 | 3.14140672 | 0.0001855 |
| 11 | 6 | 529 | 11922290683065 | 3794809718700 | 3.14173610 | -0.00014345 |

$$h_n = a_n * h_{(n-1)} + b_{(n-1)} * h_{(n-2)}$$
$$k_n = a_n * k_{(n-1)} + b_{(n-1)} * k_{(n-2)}$$

Gen CFs converge slowly

$\pi = [3; 1, \{6 | (2n+1)^2\}]$

All the mysterious and unpredictable digits of PI
come from this simple generalized CF !!

# Generalized Continued Fractions

What is another **generalized** CF encoding for $\pi$ ?

**Biersach**

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{25}{23 + \cfrac{65}{31 + \ddots}}}}$$

$\pi = [3; 1, \{(8n-1)|(8n-7)^2\}]$

There are infinitely many Generalized CFs but not all **converge**

# Generalized Continued Fractions

What is a **generalized** CF expansion for $\pi$ ?

| n | a | b | h | k | h/k | delta |
|---|---|---|---|---|---|---|
| -2 | | | 0 | 1 | | |
| -1 | 0 | 1 | 1 | 0 | | |
| 0 | 3 | 1 | 3 | 1 | 3.00000000000 | 0.14159265359 |
| 1 | 7 | 1 | 22 | 7 | 3.14285714286 | -0.00126448927 |
| 2 | 15 | 25 | 333 | 106 | 3.14150943396 | 0.00008321963 |
| 3 | 23 | 65 | 8209 | 2613 | 3.14159969384 | -0.00000704025 |
| 4 | 31 | 121 | 276124 | 87893 | 3.14159261830 | 0.00000003529 |
| 5 | 39 | 193 | 11762125 | 3744000 | 3.14159321581 | -0.00000056222 |
| 6 | 47 | 281 | 606111807 | 192931349 | 3.14159316328 | -0.00000050969 |
| 7 | 55 | 385 | 36641306510 | 11663288195 | 3.14159316801 | -0.00000051443 |
| 8 | 63 | 505 | 2541755355825 | 809065725650 | 3.14159316758 | -0.00000051399 |
| 9 | 71 | 641 | 198968490051125 | 63333627059625 | 3.14159316762 | -0.00000051403 |

$$\pi = [3; 1, \{(8n-1)|(8n-7)^2\}]$$

My GCF for $\pi$ converges faster than Euler's

# Generalized Continued Fractions

$$x = a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \ddots}}}$$

In a generalized continued fraction,
$a_n$ and $b_n$ can be any expression

$$\tan(x) = \cfrac{x}{1 - \cfrac{x^2}{3 - \cfrac{x^2}{5 - \cfrac{x^2}{7 - \ddots}}}}$$

$a_0$  $b_0$     $a_n$        $b_n$

tan(x)= [0; x, {(2n-1)|-x²}]

# Pell's Equation

- Your scientist has asked you to write a C++ program to find $x$ & $y$ for a given $n$ (assume $x, y, n \in \mathbb{Z}^+$) such that:

$$x^2 - ny^2 = 1$$

- For every $2 \leq n \leq 70$, check all $1 \leq x \leq 70{,}000$

- Why is there is no need to check for $y > \left\lceil \sqrt{\dfrac{x^2}{n}} \right\rceil$ ?

- Do you see any relationship between the specific $x$ & $y$ values that solve the equation for each $n$ value?

# Open Lab 3
## Pell's Equation

```cpp
int main()
{
    DisplayHeader();

    const uintmax_t xMax = 70000;

    for (uintmax_t n = 2; n <= 70; n++)
    {
        cout << setw(4) << n;
        bool foundSolution = false;
        uintmax_t x = 1;
        while ((x <= xMax) && !foundSolution)
        {
            uintmax_t xSqr = x * x;
            uintmax_t y = 1;
            uintmax_t yMax = sqrt(xSqr / n);
            while ((y <= yMax) && !foundSolution)
            {
                uintmax_t ySqr = y * y;
                if (xSqr - n * ySqr == 1)
                {
                    cout << setw(8) << x
                         << setw(8) << y;
                    foundSolution = true;
                }
                y++;
            }
            x++;
        }
        if (!foundSolution)
            cout << setw(8) << "-"
                 << setw(8) << "-";
        cout << endl;
    }
    return 0;
}
```

$$x^2 - ny^2 = 1$$

As soon as a valid solution is found for the current value of **n**, then stop trying any more **x** & **y** values

# Pell's Equation

Hyphens indicate no solution was found in the allowed search space

| n | x | y |
|---|---|---|
| === | ====== | ====== |
| 2 | 3 | 2 |
| 3 | 2 | 1 |
| 4 | - | - |
| 5 | 9 | 4 |
| 6 | 5 | 2 |
| 7 | 8 | 3 |
| 8 | 3 | 1 |
| 9 | - | - |
| 10 | 19 | 6 |
| 11 | 10 | 3 |
| 12 | 7 | 2 |
| 13 | 649 | 180 |
| 14 | 15 | 4 |
| 15 | 4 | 1 |
| 16 | - | - |
| 17 | 33 | 8 |
| 18 | 17 | 4 |
| 19 | 170 | 39 |
| 20 | 9 | 2 |
| 21 | 55 | 12 |
| 22 | 197 | 42 |
| 23 | 24 | 5 |

| n | x | y |
|---|---|---|
| 24 | 5 | 1 |
| 25 | - | - |
| 26 | 51 | 10 |
| 27 | 26 | 5 |
| 28 | 127 | 24 |
| 29 | 9801 | 1820 |
| 30 | 11 | 2 |
| 31 | 1520 | 273 |
| 32 | 17 | 3 |
| 33 | 23 | 4 |
| 34 | 35 | 6 |
| 35 | 6 | 1 |
| 36 | - | - |
| 37 | 73 | 12 |
| 38 | 37 | 6 |
| 39 | 25 | 4 |
| 40 | 19 | 3 |
| 41 | 2049 | 320 |
| 42 | 13 | 2 |
| 43 | 3482 | 531 |
| 44 | 199 | 30 |
| 45 | 161 | 24 |
| 46 | 24335 | 3588 |
| 47 | 48 | 7 |

| n | x | y |
|---|---|---|
| 48 | 7 | 1 |
| 49 | - | - |
| 50 | 99 | 14 |
| 51 | 50 | 7 |
| 52 | 649 | 90 |
| 53 | 66249 | 9100 |
| 54 | 485 | 66 |
| 55 | 89 | 12 |
| 56 | 15 | 2 |
| 57 | 151 | 20 |
| 58 | 19603 | 2574 |
| 59 | 530 | 69 |
| 60 | 31 | 4 |
| 61 | - | - |
| 62 | 63 | 8 |
| 63 | 8 | 1 |
| 64 | - | - |
| 65 | 129 | 16 |
| 66 | 65 | 8 |
| 67 | 48842 | 5967 |
| 68 | 33 | 4 |
| 69 | 7775 | 936 |
| 70 | 251 | 30 |

# **Check** Lab 3 – Observations

- Which values of **n** have no solution?

$$n = 1, 4, 9, 16, 25, 36, 49, \mathbf{\textcolor{red}{61}}, 64, \dots$$

- Some of the values for **x** & **y** are much bigger than for other close values of **n**:

| | | |
|---|---|---|
| 40 | 19 | 3 |
| 41 | 2049 | 320 |
| 42 | 13 | 2 |
| 43 | 3482 | 531 |
| 44 | 199 | 30 |
| 45 | 161 | 24 |
| 46 | 24335 | 3588 |
| 47 | 48 | 7 |
| 48 | 7 | 1 |

- The *magnitude* of **n** does <u>not</u> seem to be a good predictor about the magnitude of the **x** & **y** values that solve the equation for that specific **n**

# Pell's Equation: Period of Simple CF

**Small values for x & y**

| n | x | y |
|---|---|---|
| 35 | 6 | 1 |
| 47 | 48 | 7 |
| 60 | 31 | 4 |
| 68 | 33 | 4 |

Simple CF

Period = 2

$\sqrt{68} = $ {8, 4, 16, 4, 16,

**Large values for x & y**

| n | x | y |
|---|---|---|
| 13 | 649 | 180 |
| 29 | 9801 | 1820 |
| 41 | 2049 | 320 |
| 43 | 3482 | 531 |
| 46 | 24335 | 3588 |
| 53 | 66249 | 9100 |
| 61 | 1766319049 | 226153980 |
| 67 | 48842 | 5967 |

Period = 5

$\sqrt{29} = $ {5, 2, 1, 1, 2, 10, 2, 1, 1, 2, 10,

Period = 5

$\sqrt{53} = $ {7, 3, 1, 1, 3, 14, 3, 1, 1, 3, 14,

Period = 11

$\sqrt{61} = $ {7, 1, 4, 3, 1, 2, 2, 1, 3, 4, 1, 14, 1, 4, 3, 1, 2, 2, 1,

# Pell's Equation: Period of Simple CF

$$x^2 - ny^2 = 1$$

| Large values for x & y | | |
|---:|---:|---:|
| **n** | **x** | **y** |
| 13 | 649 | 180 |
| 29 | 9801 | 1820 |
| 41 | 2049 | 320 |
| 43 | 3482 | 531 |
| 46 | 24335 | 3588 |
| 53 | 66249 | 9100 |
| 61 | 1766319049 | 226153980 |
| 67 | 48842 | 5967 |

$\sqrt{13}$    **Period = 4**

$\sqrt{61}$    **Period = 11**

Simple CF

If the period of the simple CF of the $\sqrt{n}$ is large…

…then the x & y will be large for the solution to Pell's equation

# Continued Fractions

- CFs *may* have their own rich **arithmetic**, **algebra**, and potentially even their own **calculus**
  - How can one <u>directly</u> **divide** two CFs?
  - How can one <u>directly</u> take the **sin()** of a CF?
  - What does the **factorial** of a CF look like?
- In many ways a CF is a *more accurate* representation of an irrational or transcendental number
  - The sum of an infinite series must stop somewhere, and after that point, all of the remaining digits of precision are lost
  - A CF encodes the entire number **with no loss of precision**
  - What can you discover about CFs?

# Normal Distribution

- Until recently, most computer languages only provided a **uniform** pseudo-random number generator

- Growing up I had heard of a **bell curve** and I understood the rationale for **curving** test scores

- However I could not create a **normal distribution** using my 1978 vintage TRS-80 computer using Bill Gate's first **BASIC** language interpreter

*… or could I?*

# Normal Distribution

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

μ

σ

1σ

-6σ  -5σ  -4σ  -3σ  -2σ  -1σ  1σ  2σ  3σ  4σ  5σ  6σ

68.27 %    1σ

95.45 %    2σ

99.73 %    3σ

99.9937%    4σ

99.999943%    5σ

99.9999998%    6σ
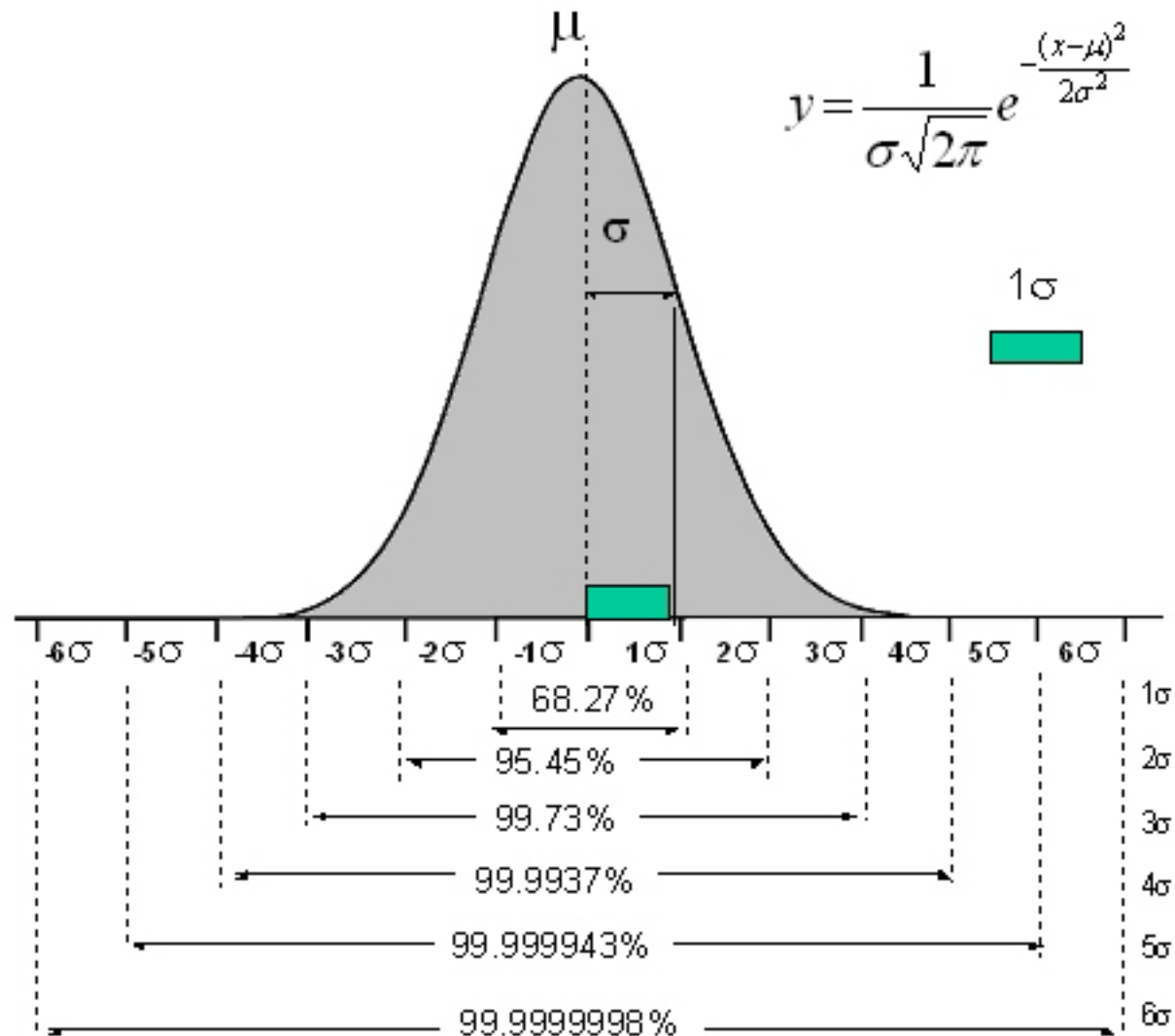
# Normal Distribution



| Sigma Number | Population Percent |
|:---:|:---:|
| 1 | 68.26% |
| 2 | 27.18% |
| 3 | 4.30% |
| 4 | 0.26% |

# Normal Distribution

- There are indeed several ways to turn a uniform distribution into a normal distribution

- Developing an accurate functional approximation to a normal curve requires advanced mathematics

- Consider this code from Abramowitz & Stegun's classic Handbook
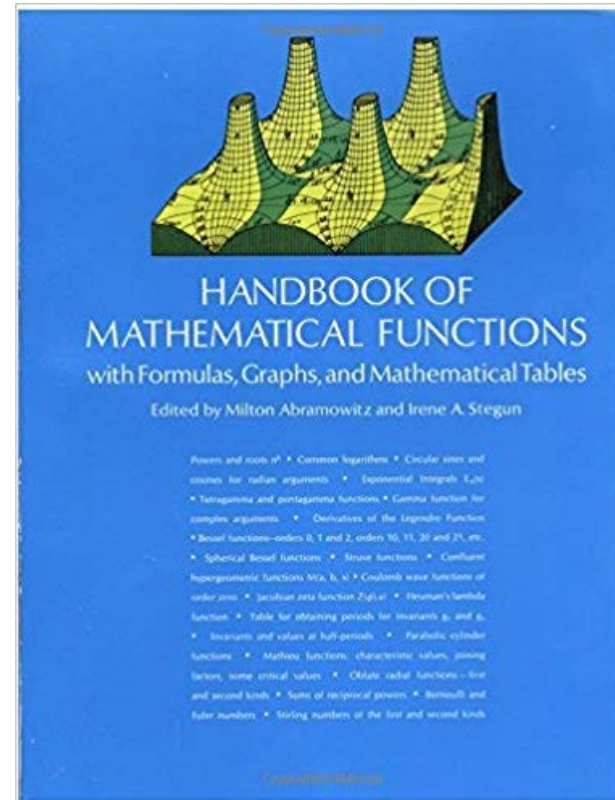
# Normal Distribution

```
double StegunNormal(double mean, double stddev)
{
    double q = 1 - distUniform(generator);
    double p = (q < 0.5) ? q : 1 - q;
    double t = sqrt(log((1 / (p * p))));
    double x = t - (2.515517 + 0.802853 * t + 0.010328 * (t * t)) /
        (1 + 1.432788 * t + 0.189269 * (t * t) + 0.001308 * (t * t * t));
    x = (q < 0.5) ? x : -1 * x;
    return x * stddev + mean;
}
```

- This was neat but I did not understand it at all!

- Where did all those magic numbers come from?

- I wanted to base my approach after something tangible

# Pachinko Distribution
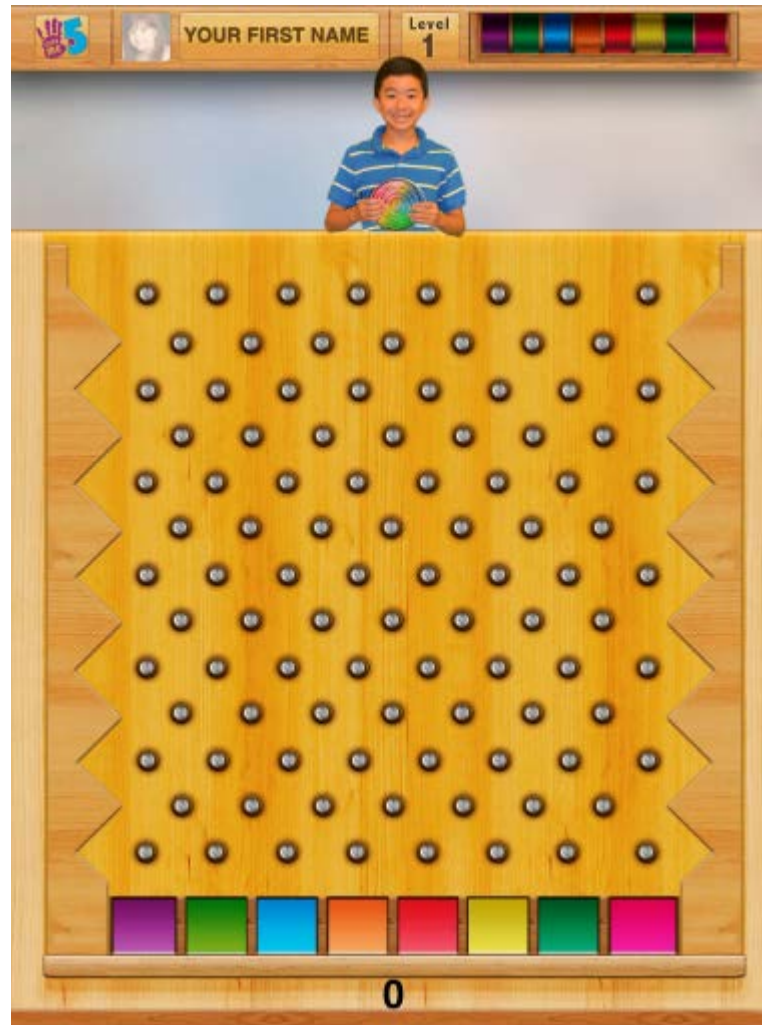
# Pachinko Distribution
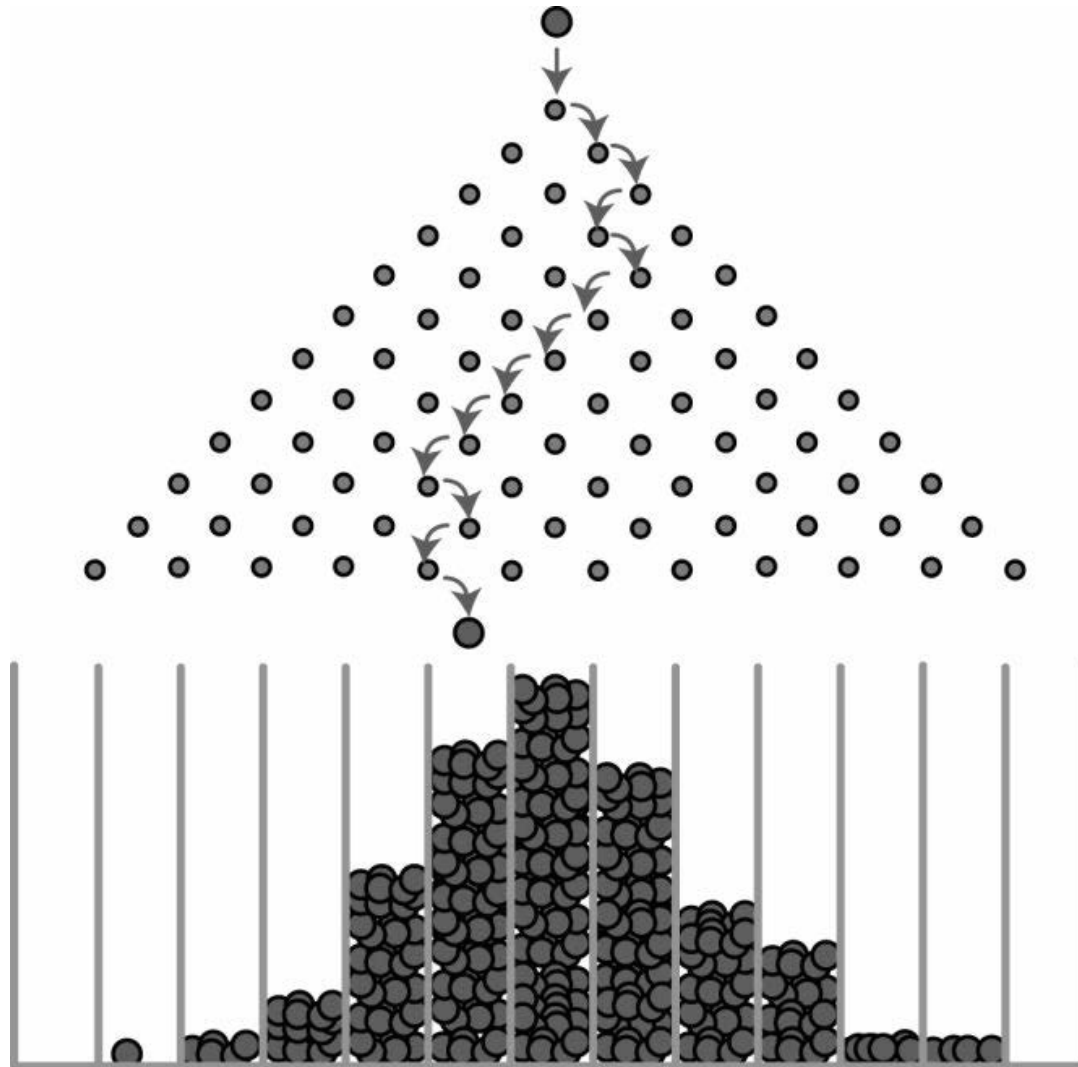
# Pachinko Distribution
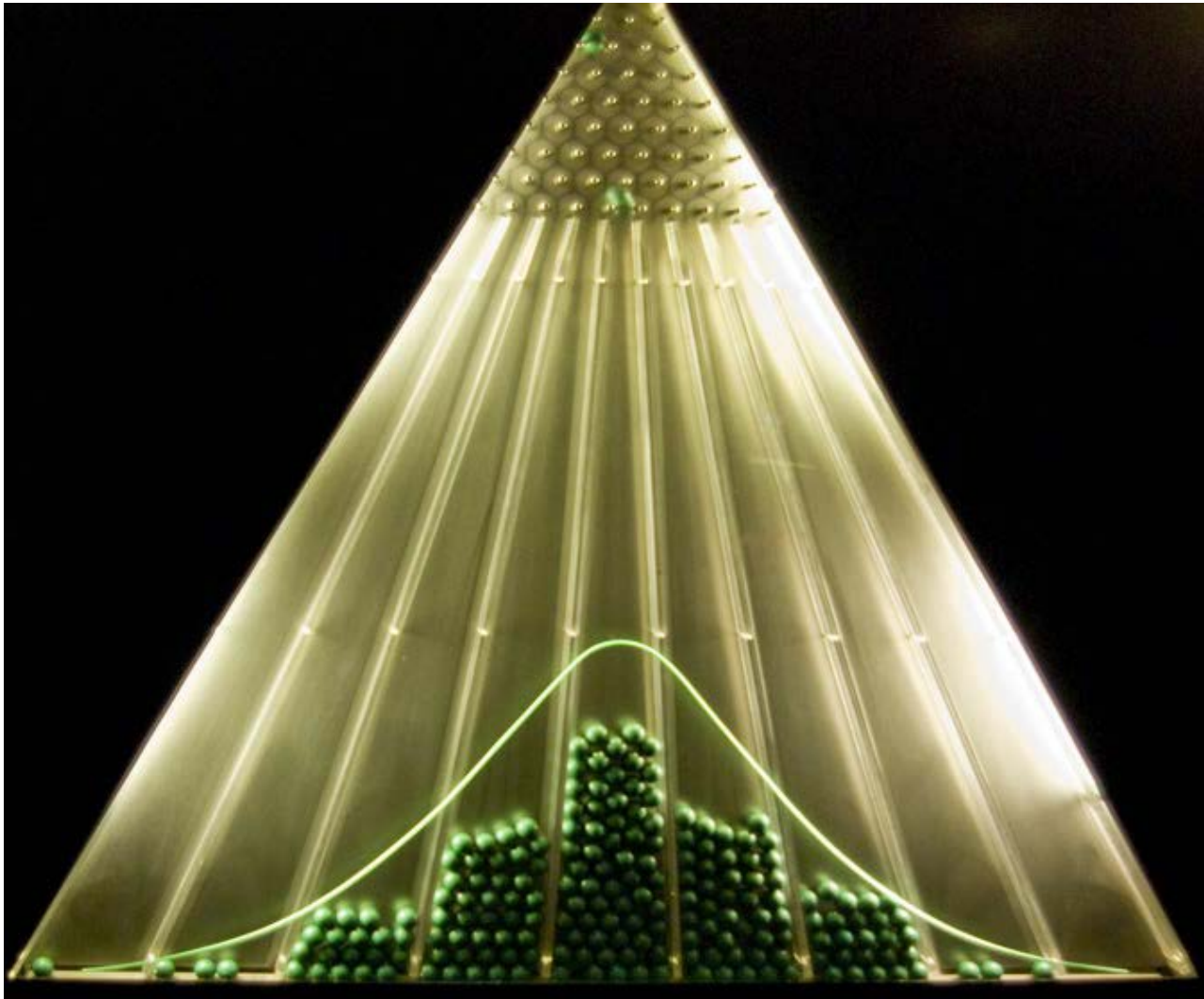
# Normal Distribution

- We can simulate dropping balls down a Pachinko board where at each **level** a ball can move one step to left or right

- If we drop enough balls through enough levels, and we accumulate the **count of balls at each slot** at the bottom, then we should be able to simulate a normal distribution

- We will run a **chi-squared test** to see if our code simulates a distribution that has a reasonable deviation from the perfect (pure) **normal distribution**

- If the discrepancies are *statistically significant*, then we cannot trust that our algorithm is producing a "good enough" normal distribution to use in scientific simulations

# **Open** Lab 4 – Pachinko Normal
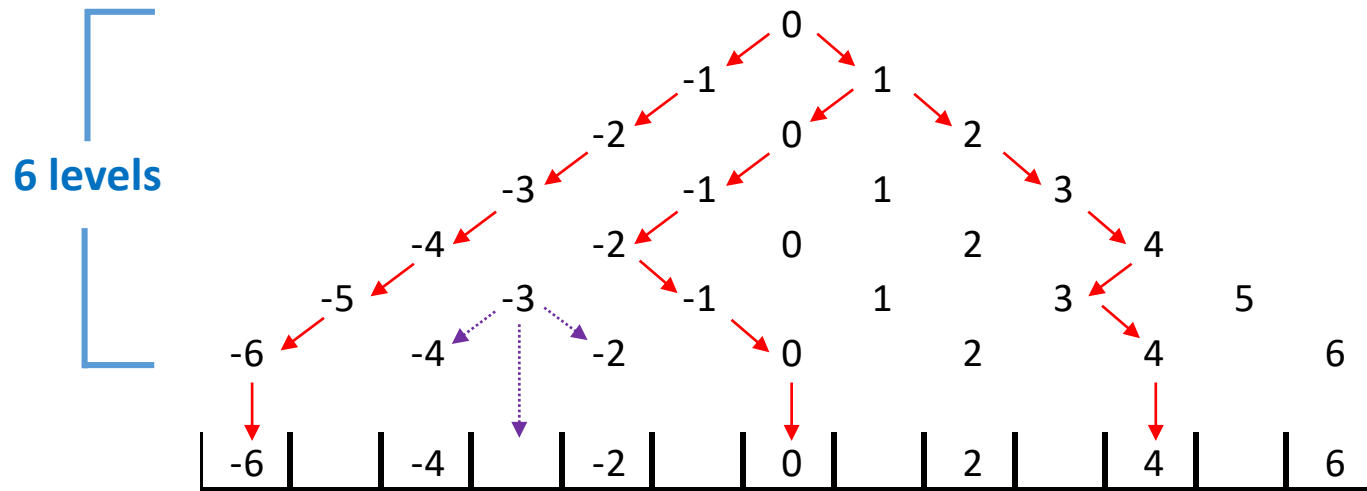
**6 levels**



The range is ½ the number of levels

```cpp
int DropBall()
{
    int slot{};
    for (int level{}; level < levels; level++) {
        int step = distribution(generator);
        if (step == 0)
            slot--;
        else
            slot++;
    }
    slot = slot / 2;
    return slot;
}
```

```cpp
seed_seq seed{ 2016 };
default_random_engine generator(seed);
uniform_int_distribution<int> distribution(0, 1);
```

# View Lab 4 – Pachinko Normal

```cpp
const int balls{ 1000 };
const int levels{ 10 };

seed_seq seed{ 2016 };
default_random_engine generator(seed);
uniform_int_distribution<int> distribution(0, 1);
double mean{};
double stddev{};

const int sigmas{ 4 };
vector<int> sigCountPachinko(sigmas);
vector<int> sigCountNormal(sigmas);

double chiSquared{};
```

```cpp
int main()
{
    CalcMeanPachinko();

    ResetPachinkoDistribution();

    CalcStdDevPachinko();

    ResetPachinkoDistribution();

    CountBallsPerSigma();

    DisplayBallsPerSigma();

    return 0;
}
```

```cpp
void ResetPachinkoDistribution()
{
    generator.seed(seed);
    distribution.reset();
}


void CalcMeanPachinko()
{
    for (int ball{}; ball < balls; ball++)
        mean += DropBall();
    mean /= balls;
}


void CalcStdDevPachinko()
{
    double variance{};
    for (int ball{}; ball < balls; ball++)
        variance += pow(DropBall() - mean, 2);
    stddev = sqrt(variance / balls);
}
```

**The mean slot *should be* = 0**
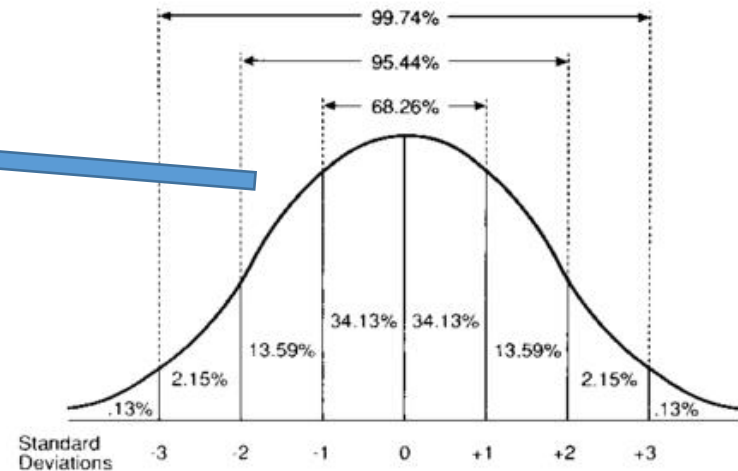
# **View** Lab 4 – Pachinko Normal

```cpp
const int balls{ 1000 };
const int levels{ 10 };
```

```cpp
const int sigmas{ 4 };
vector<int> sigCountPachinko(sigmas);
vector<int> sigCountNormal(sigmas);
```
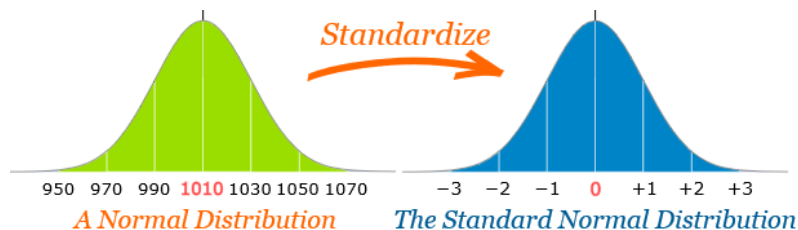
```cpp
void CountBallsPerSigma()
{
    sigCountNormal[0] = 0.6826 * balls;
    sigCountNormal[1] = 0.2718 * balls;
    sigCountNormal[2] = 0.0430 * balls;
    sigCountNormal[3] = 0.0026 * balls;

    for (int ball{}; ball < balls; ball++) {
        int slot = DropBall();
        int sigma = abs(mean + slot) / stddev;
        if (sigma < sigCountPachinko.size())
            sigCountPachinko.at(sigma)++;
    }

    for (int s{}; s < sigmas; s++)
        chiSquared += (pow(
            sigCountNormal[s] -
            sigCountPachinko[s], 2)
            / sigCountNormal.at(s));
}
```



99.74%
95.44%
68.26%
34.13% 34.13%
13.59% 13.59%
2.15% 2.15%
.13% .13%

Standard Deviations  -3  -2  -1  0  +1  +2  +3

- A **sigma** is a integral multiple of the **standard deviation**
- Each **slot** is belongs to a sigma
- We count the **number of balls** that fall into each sigma



*Standardize*

950  970  990  1010  1030  1050  1070
*A Normal Distribution*

−3  −2  −1  0  +1  +2  +3
*The Standard Normal Distribution*

57

# Chi-Squared Test

- Does the Pachinko distribution perform reasonably well compared to a perfect **normal** distribution?

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

**Karl Pearson**

```cpp
for (int s{}; s < sigmas; s++)
    chiSquared += (pow(
        sigCountNormal[s] -
        sigCountPachinko[s], 2)
        / sigCountNormal.at(s));
```

# **Run** Lab 4 – Pachinko Normal



```
                              pachinko-normal
File   Edit   View   Terminal   Tabs   Help
 Balls: 1,000
Levels: 10

    Sigma   Pachinko        Normal
        1        685           682
        2        294           271
        3         17            43
        4          4             2

Chi-Squared: 19.686
```

For 4 degrees of freedom, the **19.686** deviation is statistically *significant* (> 9.49) so the proposed algorithm **is not** generating reasonable normally distributed probabilities! ☹☹☹

| Degrees of Freedom | Probability | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.95 | 0.90 | 0.80 | 0.70 | 0.50 | 0.30 | 0.20 | 0.10 | 0.05 | 0.01 | 0.001 |
| 1 | 0.004 | 0.02 | 0.06 | 0.15 | 0.46 | 1.07 | 1.64 | 2.71 | 3.84 | 6.64 | 10.83 |
| 2 | 0.10 | 0.21 | 0.45 | 0.71 | 1.39 | 2.41 | 3.22 | 4.60 | 5.99 | 9.21 | 13.82 |
| 3 | 0.35 | 0.58 | 1.01 | 1.42 | 2.37 | 3.66 | 4.64 | 6.25 | 7.82 | 11.34 | 16.27 |
| 4 | 0.71 | 1.06 | 1.65 | 2.20 | 3.36 | 4.88 | 5.99 | 7.78 | 9.49 | 13.28 | 18.47 |
| 5 | 1.14 | 1.61 | 2.34 | 3.00 | 4.35 | 6.06 | 7.29 | 9.24 | 11.07 | 15.09 | 20.52 |
| 6 | 1.63 | 2.20 | 3.07 | 3.83 | 5.35 | 7.23 | 8.56 | 10.64 | 12.59 | 16.81 | 22.46 |
| 7 | 2.17 | 2.83 | 3.82 | 4.67 | 6.35 | 8.38 | 9.80 | 12.02 | 14.07 | 18.48 | 24.32 |
| 8 | 2.73 | 3.49 | 4.59 | 5.53 | 7.34 | 9.52 | 11.03 | 13.36 | 15.51 | 20.09 | 26.12 |
| 9 | 3.32 | 4.17 | 5.38 | 6.39 | 8.34 | 10.66 | 12.24 | 14.68 | 16.92 | 21.67 | 27.88 |
| 10 | 3.94 | 4.86 | 6.18 | 7.27 | 9.34 | 11.78 | 13.44 | 15.99 | 18.31 | 23.21 | 29.59 |
| | Nonsignificant | | | | | | | | Significant | | |

# Approaching a Normal Distribution

Maybe we didn't let enough balls drop through to get a good estimate?



What if we used 10x more balls?

# Approaching a Normal Distribution

- So can the Pachinko model accurately mimic a normal distribution?  *Yes!* …but only under the right circumstances

- It turns out that **it is not just** the **number of balls** that are used in the experiment that matters, but also the **number of levels** in the simulated Pachinko board

- The **levels** affects how wide (displacement from the center slot) a ball can fall left or right from the topmost (first) pin

- We have to ensure we have enough **levels** (therefore enough **width** at the *bottom* level) to ensure the balls can **spread out** during their fall to occupy the side slots that represent the **higher sigma values**

# Approaching a Normal Distribution

Increasing the **number of levels** improves the agreement of the **sigma ball count** between the Pachinko and perfect normal distribution, **thereby decreasing the chi-squared value**, until the difference **is no longer statistically significant**

| Degrees of Freedom | Probability | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.95 | 0.90 | 0.80 | 0.70 | 0.50 | 0.30 | 0.20 | 0.10 | 0.05 | 0.01 | 0.001 |
| 1 | 0.004 | 0.02 | 0.06 | 0.15 | 0.46 | 1.07 | 1.64 | 2.71 | 3.84 | 6.64 | 10.83 |
| 2 | 0.10 | 0.21 | 0.45 | 0.71 | 1.39 | 2.41 | 3.22 | 4.60 | 5.99 | 9.21 | 13.82 |
| 3 | 0.35 | 0.58 | 1.01 | 1.42 | 2.37 | 3.66 | 4.64 | 6.25 | 7.82 | 11.34 | 16.27 |
| 4 | 0.71 | 1.06 | 1.65 | 2.20 | 3.36 | 4.88 | 5.99 | 7.78 | 9.49 | 13.28 | 18.47 |
| 5 | 1.14 | 1.61 | 2.34 | 3.00 | 4.35 | 6.06 | 7.29 | 9.24 | 11.07 | 15.09 | 20.52 |
| 6 | 1.63 | 2.20 | 3.07 | 3.83 | 5.35 | 7.23 | 8.56 | 10.64 | 12.59 | 16.81 | 22.46 |
| 7 | 2.17 | 2.83 | 3.82 | 4.67 | 6.35 | 8.38 | 9.80 | 12.02 | 14.07 | 18.48 | 24.32 |
| 8 | 2.73 | 3.49 | 4.59 | 5.53 | 7.34 | 9.52 | 11.03 | 13.36 | 15.51 | 20.09 | 26.12 |
| 9 | 3.32 | 4.17 | 5.38 | 6.39 | 8.34 | 10.66 | 12.24 | 14.68 | 16.92 | 21.67 | 27.88 |
| 10 | 3.94 | 4.86 | 6.18 | 7.27 | 9.34 | 11.78 | 13.44 | 15.99 | 18.31 | 23.21 | 29.59 |
| | Nonsignificant | | | | | | | | Significant | | |

```
                    pachinko-normal
 File  Edit  View  Terminal  Tabs  Help
 Balls: 1,000
 Levels: 10

    Sigma   Pachinko      Normal
      1        685           682
      2        294           271
      3         17            43
      4          4             2

 Chi-Squared: 19.686


 Process returned 0 (0x0)    exec
 Press ENTER to continue.
```

```
                    pachinko-normal
 File  Edit  View  Terminal  Tabs  Help
 Balls: 1,000
 Levels: 100

    Sigma   Pachinko      Normal
      1        644           682
      2        301           271
      3         52            43
      4          3             2

 Chi-Squared: 7.822


 Process returned 0 (0x0)    exec
 Press ENTER to continue.
```

```
                    pachinko-normal
 File  Edit  View  Terminal  Tabs  Help
 Balls: 1,000
 Levels: 1,000

    Sigma   Pachinko      Normal
      1        662           682
      2        295           271
      3         41            43
      4          2             2

 Chi-Squared: 2.805


 Process returned 0 (0x0)    exec
 Press ENTER to continue.
```

# Central Limit Theorem

## Central limit theorem

In probability theory, the **central limit theorem** (**CLT**) establishes that, for the most commonly studied scenarios, when independent random variables are added, their sum tends toward a normal distribution (commonly known as a *bell curve*) even if the original variables themselves are not normally distributed. In more precise terms, given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined (finite) expected value and finite variance, will be approximately normally distributed, regardless of the underlying distribution.[1][2] The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.
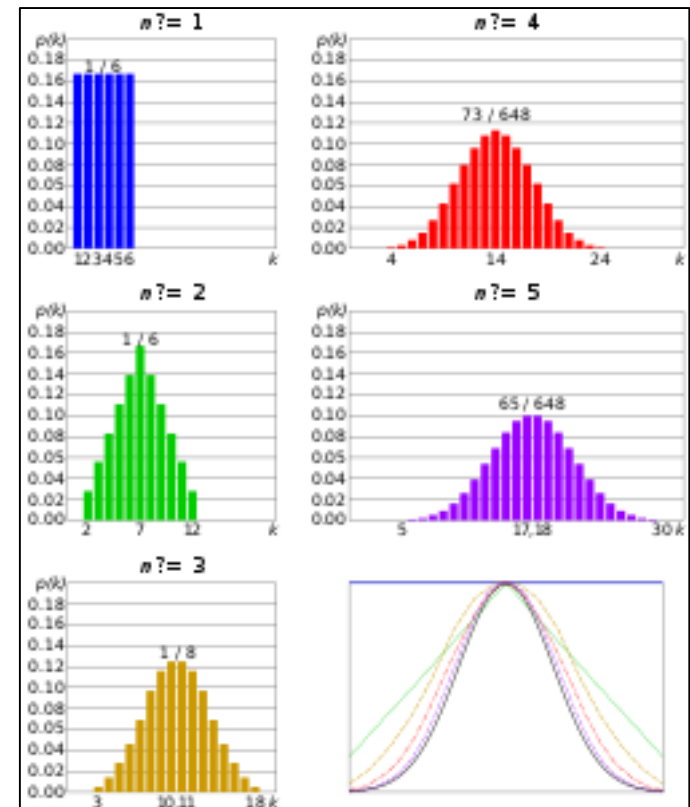
# Approximating a Normal Distribution

It is best to use the code from Abramowitz & Stegun, as only **one** call to this function is required to produce a normalized random variable versus to my method:

(1000 balls x 1000 levels = **1M** calls!)
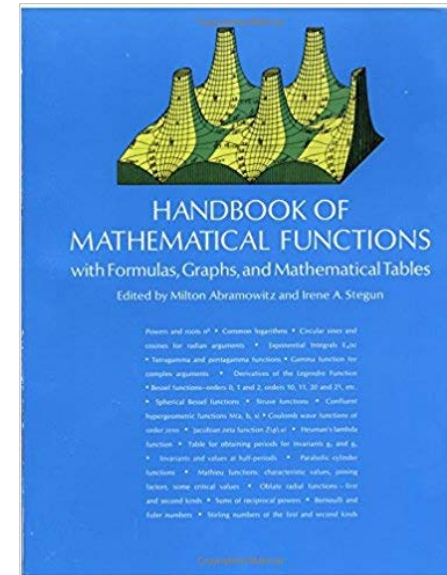
```
double StegunNormal(double mean, double stddev)
{
    double q = 1 - distUniform(generator);
    double p = (q < 0.5) ? q : 1 - q;
    double t = sqrt(log((1 / (p * p))));
    double x = t - (2.515517 + 0.802853 * t + 0.010328 * (t * t)) /
        (1 + 1.432788 * t + 0.189269 * (t * t) + 0.001308 * (t * t * t));
    x = (q < 0.5) ? x : -1 * x;
    return x * stddev + mean;
}
```

# The C++ has a built-in **normal_distribution()**

```cpp
int main()
{
    std::random_device rd{};
    std::mt19937 gen{rd()};

    // values near the mean are the most likely
    // standard deviation affects the dispersion of generated values from the mean
    std::normal_distribution<> d{5,2};

    std::map<int, int> hist{};
    for(int n=0; n<10000; ++n) {
        ++hist[std::round(d(gen))];
    }
    for(auto p : hist) {
        std::cout << std::setw(2)
                  << p.first << ' ' << std::string(p.second/200, '*') << '\n';
    }
}
```

# Now you know...

- Rational, Irrational, and Transcendental numbers each have their own style of continued fractions

  - We can take any real number and *generate* a CF

  - Given a CF, we can *expand* it to regain the original number

- The **convergents** of a CF are excellent approximations to the original number

- The magnitude of the **x** & **y** values in solutions to **Pell's Equation** $\{x^2 - ny^2 = 1\}$ is related to the <u>period</u> of the *simple* continued fraction of $\sqrt{n}$

- Memorizing thousands of digits of $\pi$ is okay – but I'd rather appreciate its beautifully simple GCF:  **[3; 1, {6|(2n+1)²}]**

# Now you know…

- A perfect **Normal distribution** ensures that **68.26%** of all values fall within **one** (1) standard deviation from the **mean**

    - 99.73% of all values in a perfect normal distribution are within **three (3)** standard deviations from the mean

    - The normal distribution is known as the "**bell curve**"

- There is a way to convert a PRNG created uniform distribution into a normal distribution – but **don't use the Pachinko method**

- The **chi-squared** test suggests if the discrepancies between the observed and the expected values are <u>statistically significant</u>