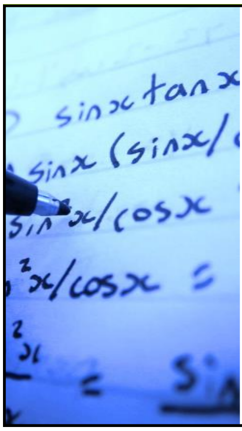


Numerical Methods

Optional events

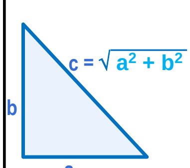
Individual Test (topics 1-8)	2023-12-21
Team Contest (topics 1-11)	2024-01-11

Tentative schedule of lectures

No.	Topic	Date
1	Imperative Programming	2023-10-09
2	Digital Circuits	2023-10-16
3	Computers	2023-10-23
4	Subprograms	2023-11-06
5	Text Processing	2023-11-13
6	Object-oriented Programming	2023-11-20
7	Numerical methods	2023-11-27
8	Computational Complexity	2023-12-04
9	Databases and Machine Learning	2023-12-11
10	Parallel Processing	2023-12-18
11	Computer Networks & Cybersecurit	2024-01-08
12	Software Engineering	2024-01-15
13	Embedded Systems	2024-01-22
14	Professionalism in Computing	2024-01-29

Aim of the lecture

Length of hypotenuse



$a^2 + b^2 = a^2 (1 + b^2/a^2)$

$a^2 + b^2 = a^2 (1 + (b/a)^2)$

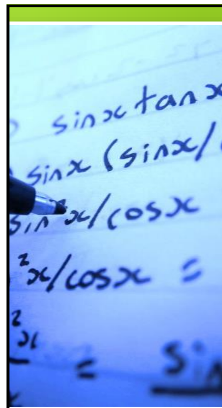
$\sqrt{a^2 + b^2} = \sqrt{a^2}\ \sqrt{1 + (b/a)\ (b/a)}$

$\sqrt{a^2 + b^2} = a\sqrt{1 + (b/a)\ (b/a)}$

Aim of the lecture

Aim of the lecture

Agenda



Integers

Introduction to real numbers

C

`float`

Format specifiers:

`%f`

`%g`

Simple programs

C

```
#include <stdio.h>
int main(){
    float  x= 0.4, y= 7.5;
    printf("%f\n", x*y); }
```

```
3.000000
```

C

```
#include <stdio.h>
int main(){
    float  x= 0.4, y= 7.5;
    printf("%g\n", x*y); }
```

```
3
```


Simple programs

C

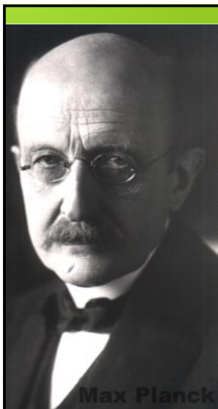
```
#include <stdio.h>
int main(){
    float  x= 0.4, y= 7.5;
    printf("%f\n", x*y); }
```

```
3.000000
```

```
x= 0.4
y= 7.5
print(x*y)
```

```
3.0
```

Scientific notation



$6.626 \cdot 10^{-34}$

Scientific notation


```c
#include <stdio.h>
int main(){
    float x= 6.626*10-34;
    printf("%f\n", x);}
```

```
32.259998
```

```python
x= 6.626*10-34
print(x)
```

```
32.260000000000005
```


E notation

$a\,\mathrm{e}\,b \equiv a \cdot 10^{b}$

5 e -2 == $5*10^{-2}$ == 0.05
5 e 0 == $5*10^{0}$ == 5
5 e 2 == $5*10^{2}$ == 500

Scientific notation


```c
#include <stdio.h>
int main(){
    float x= 6.626e-34;
    printf("%g\n", x);}
```

```
6.626e-34
```

```python
x= 6.626e-34
print(x)
```

```
6.626e-34
```


Mathematical library

```c
#include <stdio.h>
#include <math.h>
int main(){
    float a= 3e-1;
    float b= 4e-1;
    float r= sqrt(a*a + b*b);
    printf("%g\n", r); }
```

```
0.5
```

```python
import math

a= 3e-1
b= 4e-1
r= math.sqrt(a*a + b*b)
print(r)
```

```
0.5
```


Type of result of arithmetical operators

```c
#include <stdio.h>
int main(){
    float a= 10.0 / 4;
    float b= 10   / 4;
    printf("%g %g\n", a, b);}
```

```
2.5 2
```

```python
a= 10.0 / 4
b= 10   / 4
print(a, b)
```

```
2.5 2.5
```

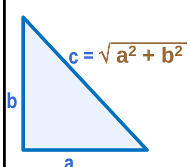
Real numbers in C

C

```
float
double
```

Real numbers

Length of hypotenuse



$\sqrt{a^2 + b^2} = a\sqrt{1 + (b/a)\,(b/a)}$

The hypotenuse problem

$c\ \sqrt{a^2 + b^2} = c\ a\sqrt{1 + (b/a)\,(b/a)}$

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
     double m;
     m= C*sqrt(A*A + B*B);
     printf("m= %g\n",m);
     return(0);
     }
```

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
     double m;
     m= C*A*sqrt(1 + (B/A)*(B/A));
     printf("m= %g\n",m);
     return(0);
     }
```


... but they are not

$c\ \sqrt{a^2 + b^2} = c\ a\sqrt{1 + (b/a)\,(b/a)}$

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
     double m;
     m= C*sqrt(A*A + B*B);
     printf("m= %g\n",m);
     return(0);
     }
```

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
     double m;
     m= C*A*sqrt(1 + (B/A)*(B/A));
     printf("m= %g\n",m);
     return(0);
     }
```

m= 0 ≠ m= 5

Python version

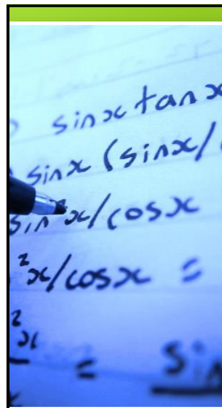
$c\ \sqrt{a^2 + b^2} = c\ a\sqrt{1 + (b/a)\,(b/a)}$

```
import math
A= 3e-200
B= 4e-200
C= 1e200
m= C*math.sqrt(A*A + B*B)
print("m=", m)
```

```
import math
A= 3e-200
B= 4e-200
C= 1e200
m= C*A*math.sqrt(1 + (B/A)*(B/A))
print("m=", m)
```

m= 0.0 ≠ m= 5.0

Agenda



What will the result be: 'F' or 'Finished'?

C

```
#include <stdio.h>
int main(){
    float  x;
    printf("F");
    for(x= 1.0; x > 0.0; x/= 10)
        ;
    printf("inished\n");

    }
```

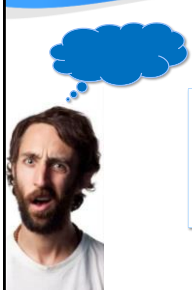
Finished

What will the result be: 'F' or 'Finished'?

```
print("F", end="")
x= 1.0
while x > 0.0:
      x= x / 10
print("inished")
```

Finished

Question



C

```
#include <stdio.h>
int main(){
    float  x;
    printf("F");
    for(x= 1.0; x > 0.0; x/= 10)
        ;
    printf("inished\n"); }
```

Finished

How real numbers might be represented?

0	0	1	0	1	1	0	0

Integer part / Fraction part

How real numbers might be represented?

0	0	1	0	1	1	0	0

$$\text{Value} = 0\cdot2^{3} + 0\cdot2^{2} + 1\cdot2^{1} + 0\cdot2^{0} + 1\cdot2^{-1} + 1\cdot2^{-2} + 0\cdot2^{-3} + 0\cdot2^{-4}$$

$$= 2 + 0.75$$

How real numbers might be represented?

s	a_3	a_2	a_1	a_0	b_1	b_2	b_3	b_4

$$(-1)^{s}\left(a_3\cdot2^{3} + a_2\cdot2^{2} + a_1\cdot2^{1} + a_0\cdot2^{0} + b_1\cdot2^{-1} + b_2\cdot2^{-2} + b_3\cdot2^{-3} + b_4\cdot2^{-4}\right)$$

Real numbers in C

C

$$\left.\begin{array}{l}\texttt{float}\\ \texttt{double}\end{array}\right\}\text{Real numbers}$$

Puzzle

```
#include <stdio.h>
int main(){
    float  x;
    double y;
    printf("%d, %d\n", sizeof x, sizeof y);
    }
```

C

```
4, 8
```

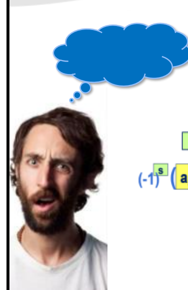

Question

```
#include <stdio.h>
int main(){
    float  x;
    printf("F");
    for(x= 1.0; x > 0.0; x/= 10)
        ;
    printf("inished\n");
    }
```

C

```
Finished
```


Question



s	a_3	a_2	a_1	a_0	b_1	b_2	b_3	b_4

$$(-1)^{s}\left(a_3\cdot2^{3} + a_2\cdot2^{2} + a_1\cdot2^{1} + a_0\cdot2^{0} + b_1\cdot2^{-1} + b_2\cdot2^{-2} + b_3\cdot2^{-3} + b_4\cdot2^{-4}\right)$$

Union type

```
union  Point1{
       float x;
       float y; };
struct Point2{
       float x;
       float y; };
union  Point1 a;
struct Point2 b;
}
```


Puzzle

```
#include <stdio.h>
int main(){
    union  Point1{
           float x;
           float y; };
    struct Point2{
           float x;
           float y; };
    union  Point1 a;
    struct Point2 b;
    printf("%d\n", sizeof (float));
    printf("%d, %d\n", sizeof a, sizeof b);}
```

```
4
4, 8
```


Union type

```
union  Point1{
       float x;
       float y; };
struct Point2{
       float x;
       float y; };
union  Point1 a;
struct Point2 b;
}
```


Internal representation of the float type

```
#include <stdio.h>
int main(){
    union Real {
          float x;
          unsigned char y[4];
    };
    union Real z;
    z.x= 4.0;
    printf("%g\n", z.x);
    printf("%x,%x,%x,%x\n", z.y[0],z.y[1],z.y[2],z.y[3]);
    }
```

```
4
0,0,80,40
```


How real numbers might be represented?

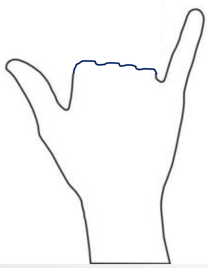
0	1	0	0	0	0	0	0

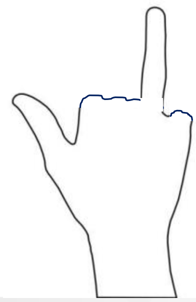
Integer part Fraction part

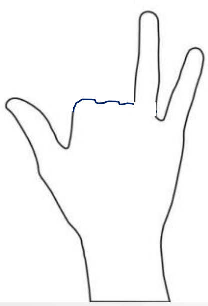
Fixed-point representation – Weakness

≠ 0 ≠ 0

0	1	0	1	0	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	0	1







Normalization

0	1	0	1	0	0	0	0	0
0	0	0	0		0	1	0	0

Normalization

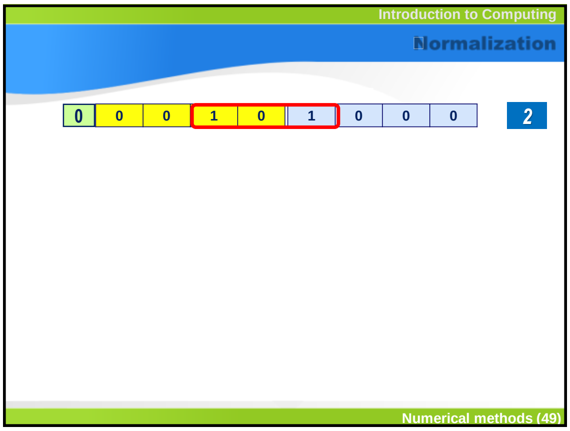
0	1	0	1	0	0	0	0	0

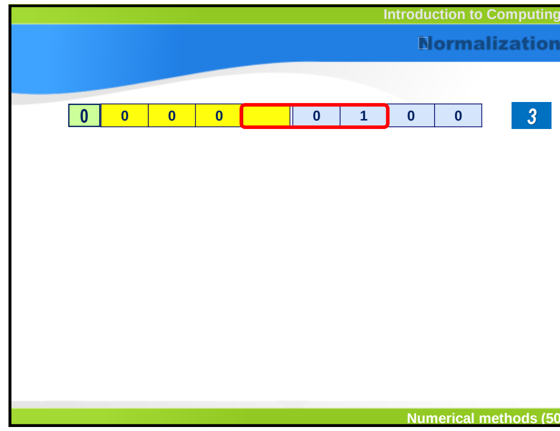
0

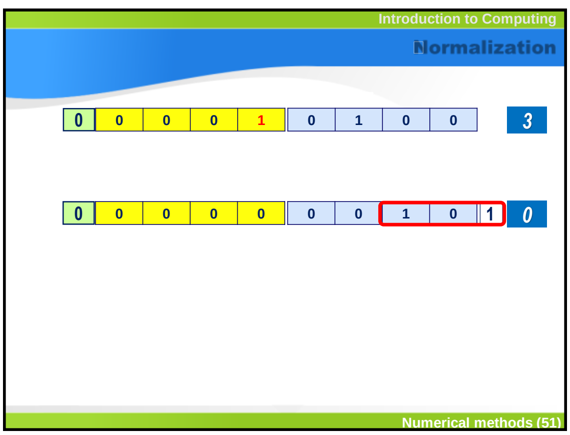
Normalization

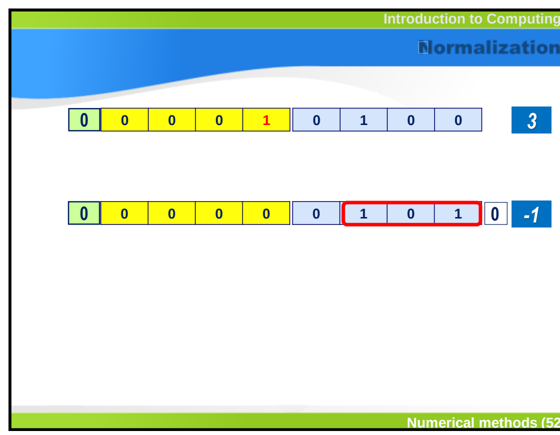
0	0	1	0	1	0	0	0	0

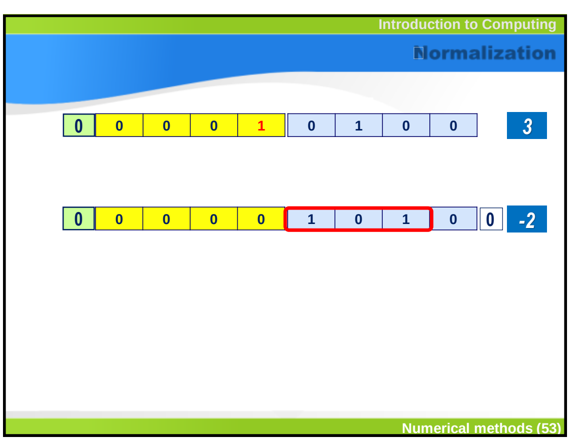
1

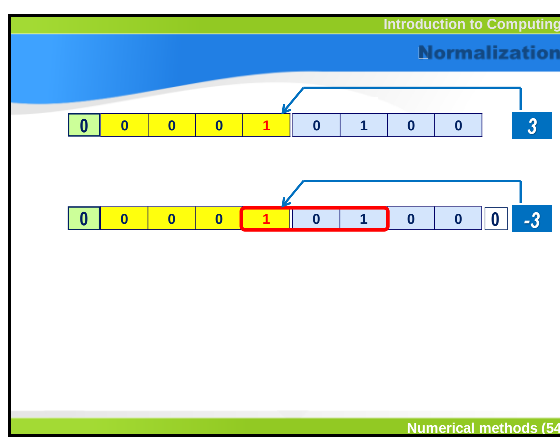

Introduction to Computing
Normalization
0 0 0 1 0 1 0 0 0
2
Numerical methods (49)


Introduction to Computing
Normalization
0 0 0 0 0 1 0 0
3
Numerical methods (50)


Introduction to Computing
Normalization
0 0 0 0 1 0 1 0 0
3
0 0 0 0 0 0 0 1 0 1
0
Numerical methods (51)


Introduction to Computing
Normalization
0 0 0 0 1 0 1 0 0
3
0 0 0 0 0 0 1 0 1 0
-1
Numerical methods (52)


Introduction to Computing
Normalization
0 0 0 0 1 0 1 0 0
3
0 0 0 0 0 1 0 1 0 0
-2
Numerical methods (53)


Introduction to Computing
Normalization
0 0 0 0 1 0 1 0 0
3
0 0 0 0 1 0 1 0 0 0
-3
Numerical methods (54)

Floating point representation

0	0	1	0	0	1	1	1	0

0	1	1	1	0	1	1	1	0
s	e				m			

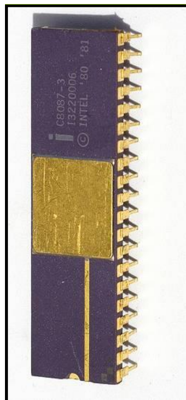
$x \neq 0$: $x = (-1)^s \cdot 2^e \cdot m$

$s \in \{0, 1\}$

e= exponent

m= mantissa $\in [1, 2)$

IEEE 754-1985



level	width	range
single precision	32 bits	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
double precision	64 bits	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$

http://www.flickr.com/photos/brit_robin/5574888096/sizes/m/in/photostream/

IEEE 754-1985: Single precision

sign exponent (8-bit) fraction (23-bit)

0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

31 23 0

$0.15625_{10} = 1/8 + 1/32 = 0.00101_2 = 1.01_2 \bullet 2^{-3}$

$.01_2$ -3

$127 + (-3) = 124$

http://en.wikipedia.org/wiki/IEEE_754-1985

IEEE 754-1985

sign exponent (8-bit) fraction (23-bit)

0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

31 23 0

$0.15625_{10} = 1/8 + 1/32 = 0.00101_2 = 1.01_2 \bullet 2^{-3}$

$.01_2$ -3

$127 + (-3) = 124$

Internal representation of the `float` type

C

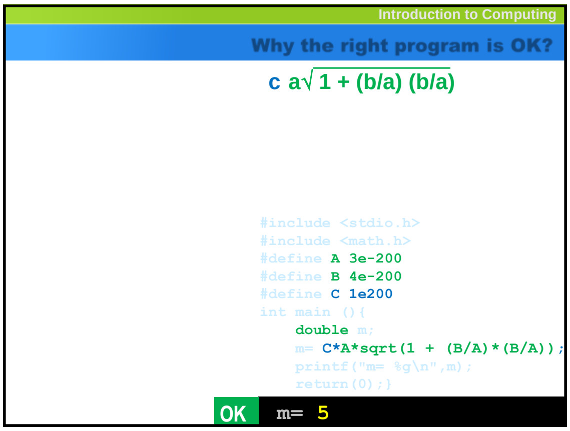
```c
#include <stdio.h>
int main(){
    union Real {
          float x;
          unsigned char y[4];
    };
    union Real z;
    z.x= 0.15625;
    printf("%g\n", z.x);
    printf("%x,%x,%x,%x\n", z.y[0],z.y[1],z.y[2],z.y[3]);
    }
```

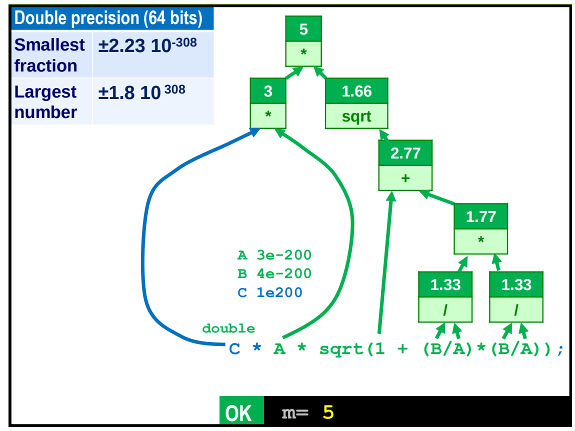
```
0.15625
0,0,20,3e
```

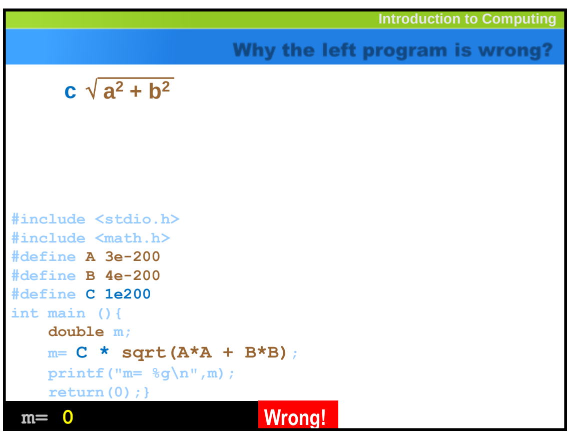

IEEE 754-1985

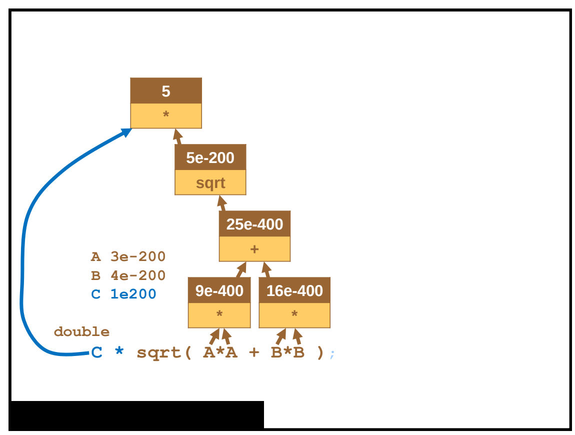
0 0 1 1 | 1 1 1 0 | 0 0 1 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0

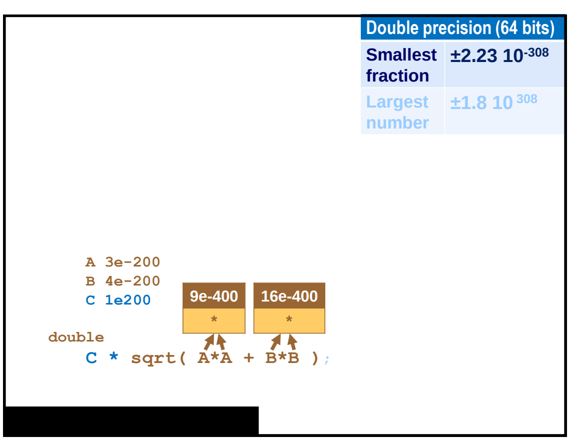
```
0.15625
0,0,20,3e
```

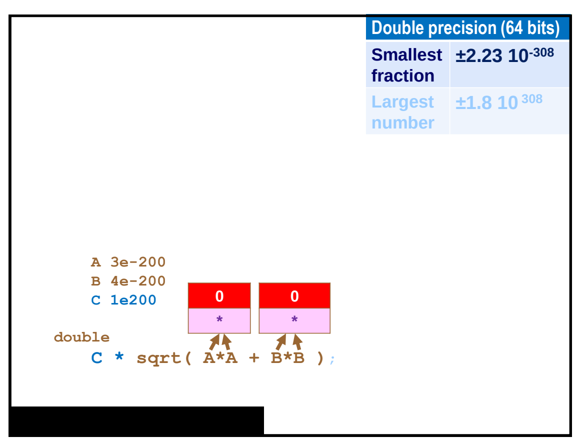

Introduction to Computing
Why the right program is OK?
c a√ 1 + (b/a) (b/a)
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
double m;
m= C*A*sqrt(1 + (B/A)*(B/A));
printf("m= %g\n",m);
return(0);}
OK
m= 5


Double precision (64 bits)
Smallest fraction ±2.23 10-308
Largest number ±1.8 10 308
5
*
3
*
1.66
sqrt
2.77
+
1.77
*
1.33
/
1.33
/
A 3e-200
B 4e-200
C 1e200
double
C * A * sqrt(1 + (B/A)*(B/A));
OK
m= 5

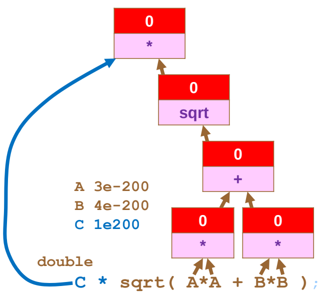

Introduction to Computing
Why the left program is wrong?
c √ a2 + b2
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
double m;
m= C * sqrt(A*A + B*B);
printf("m= %g\n",m);
return(0);}
m= 0
Wrong!


5
*
5e-200
sqrt
25e-400
+
9e-400
*
16e-400
*
A 3e-200
B 4e-200
C 1e200
double
C * sqrt(A*A + B*B);

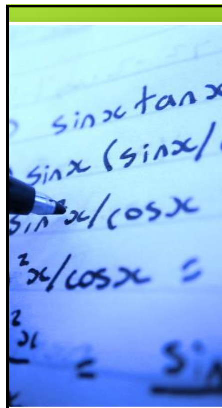

Double precision (64 bits)
Smallest fraction ±2.23 10-308
Largest number ±1.8 10 308
A 3e-200
B 4e-200
C 1e200
9e-400
*
16e-400
*
double
C * sqrt(A*A + B*B);


Double precision (64 bits)
Smallest fraction ±2.23 10-308
Largest number ±1.8 10 308
A 3e-200
B 4e-200
C 1e200
0
*
0
*
double
C * sqrt(A*A + B*B);

Double precision (64 bits)	
Smallest fraction	±2.23 10^{-308}
Largest number	±1.8 10^{308}

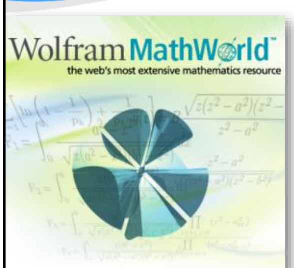


m= 0 Wrong!



Agenda

Numerical stability



In a numerically stable algorithm, errors in the input lessen in significance as the algorithm executes, having little effect on the final output.

Macura, Wiktor K. "Numerical Stability." From *MathWorld*--A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld.wolfram.com/NumericalStability.html

---

Which one is numerically stable?

$c \cdot \sqrt{a^2 + b^2}$ = $c \cdot a\sqrt{1 + (b/a)\cdot(b/a)}$

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
    double m;
    m= C*sqrt(A*A + B*B);
    printf("m= %g\n",m);
    return(0);
    }
```

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
    double m;
    m= C*A*sqrt(1 + (B/A)*(B/A));
    printf("m= %g\n",m);
    return(0);
    }
```

m= 0 ≠ m= 5

Numerical stability

$c\ a\sqrt{1 + (b/a)\ (b/a)}$

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
    double m;
    m=C*A*sqrt(1+(B/A)*(B/A));
    printf("m= %g\n",m);
    return(0);}
```

Numerical stability

$c\ a\sqrt{1 + (b/a)\ (b/a)}$

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e+200
#define C 1e2
int main (){
    double m;
    m=C*A*sqrt(1+(B/A)*(B/A));
    printf("m= %g\n",m);
    return(0);}
```

Numerical stability

c a√ 1 + (b/a) (b/a)

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e+200
#define C 1e2
int main (){
    double m;
    m=C*A*sqrt(1+(B/A)*(B/A));
    printf("m= %g\n",m);
    return(0);}
```

∞ ∞

1+ ∞

√ ∞

∞

Double precision (64 bits)	
Smallest fraction	$\pm 2.23 \; 10^{-308}$
Largest number	$\pm 1.8 \; 10^{308}$

An alternative solution

c b√(a/b) (a/b) + 1 = c a√ 1 + (b/a) (b/a)

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e+200
#define C 1e2
int main (){
    double m;
    m=C*B*sqrt((A/B)*(A/B)+1);
    printf("m= %g\n",m);
    return(0);}
```

0.75e-400

Double precision (64 bits)	
Smallest fraction	$\pm 2.23 \; 10^{-308}$
Largest number	$\pm 1.8 \; 10^{308}$

An alternative solution

c b√(a/b) (a/b) + 1 = c a√ 1 + (b/a) (b/a)

4e+202 * 1.0 = 4e+202

√1.0

0.0 +1

0.0 * 0.0

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e+200
#define C 1e2
int main (){
    double m;
    m=C*B*sqrt((A/B)*(A/B)+1);
    printf("m= %g\n",m);
    return(0);}
```

Double precision (64 bits)	
Smallest fraction	$\pm 2.23 \; 10^{-308}$
Largest number	$\pm 1.8 \; 10^{308}$

What to choose?

a	b	b√(a/b) (a/b) + 1	a√ 1 + (b/a) (b/a)
Large	Large		
Large	*small*		
small	Large		
small	*small*		

What to choose?

a	b	b√(a/b) (a/b) + 1	a√ 1 + (b/a) (b/a)
Large	Large		
Large a > b *small*			●
small a < b Large		●	
small	*small*		

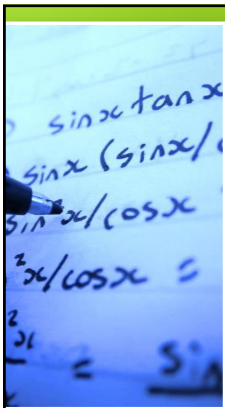
Numerical stability

c a√ 1 + (b/a) (b/a) =

= c b√(a/b) (a/b) + 1

```
#include <stdio.h>
#include <math.h>
#define A 3e-200
#define B 4e-200
#define C 1e200
int main (){
    double m;
    if (A > B)
      m=C*A*sqrt(1+(B/A)*(B/A));
    else
      m=C*B*sqrt((A/B)*(A/B)+1);
    printf("m= %g\n",m);
    return(0);}
```

Numerically stable



Agenda

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

$$p(x) = 1 + 2 \cdot x + 3 \cdot x^2$$

$$p(2) =$$

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
```


Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```
#include <stdio.h>
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
 void main(){
      double a[]= {0, 0, 0, 0, 0, 0, 0, 0, 1e300};
      printf("%g\n", p(1e-50, 8, a));
      return; }
```

$$p(x) = 10^{300} x^8$$

$$p(10^{-50}) =$$

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```
#include <stdio.h>
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
 void main(){
      double a[]= {0, 0, 0, 0, 0, 0, 0, 0, 1e300};
      printf("%g\n", p(1e-50, 8, a));
      return; }
```

$$p(x) = 10^{300} x^8$$

$$p(10^{-50}) = 10^{300}\ 10^{-400}$$

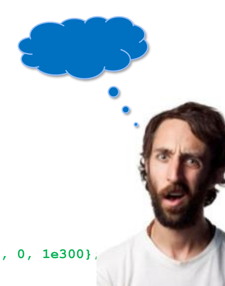
$$p(10^{-50}) = 10^{-100}$$

0

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```
#include <stdio.h>
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
 void main(){
      double a[]= {0, 0, 0, 0, 0, 0, 0, 0, 1e300};
      printf("%g\n", p(1e-50, 8, a));
      return; }
```

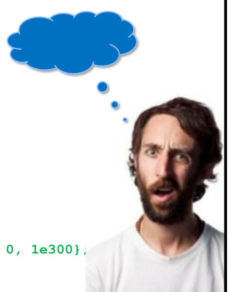


0

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```c
#include <stdio.h>
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
 void main(){
      double a[]= {0, 0, 0, 0, 0, 0, 0, 0, 1e300};
      printf("%g\n", p(1e-50, 8, a));
      return; }
```



0

Why?

Double precision (64 bits)	
Smallest fraction	$\pm 2.23\ 10^{-308}$

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```c
#include <stdio.h>
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
 void main(){
      double a[]= {0, 0, 0, 0, 0, 0, 0, 0, 1e300};
      printf("%g\n", p(1e-50, 8, a));
      return; }
```

$p(x) = 10^{300}\, x^8$

$p(10^{-50}) = 10^{300}\, 10^{-400}$

$p(10^{-50}) = 10^{-100}$

0

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```c
#include <stdio.h>
double Power(double b, int k){
      double res= 1.0;
      int i;
      for (i= 1; i <= k; i++)
          res*= b;
      return res; }
double p(double x, int n, double a[]){
      double result = 0.0;
      int k;
      for (k= 0; k <= n; k++)
          result += a[k]*Power(x,k);
      return result; }
 void main(){
      double a[]= {0, 0, 0, 0, 0, 0, 0, 0, 1e300};
      printf("%g\n", p(1e-50, 8, a));
      return; }
```

$p(x) = 10^{300}\, x^8$

$p(10^{-50}) = 10^{300}\, 10^{-400}$

$p(10^{-50}) = 10^{-100}$

Unstable

0

Polynomial with the Power function

$$p(x) = \sum_{k=0}^{n} a_k x^k = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots$$

```python
def Power(b, k):
    res= 1.0
    for i in range(1, k+1):
        res*= b
    return res
def p(x, n, a):
    result= 0.0
    for k in range(0, n+1):
        result+= a[k]*Power(x, k)
    return result
a=[0, 0, 0, 0, 0, 0, 0, 0, 1e300]
print(p(1e-50, 8, a))
```

$p(x) = 10^{300}\, x^8$

$p(10^{-50}) = 10^{300}\, 10^{-400}$

$p(10^{-50}) = 10^{-100}$

Unstable

0.0

Horner scheme

William George Horner

1786 – 1837

Horner scheme

Theorem. Value of a polynomial

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n$$

can be computed using the following recursive function:

$p(x,0) = a_0$

$p(x,n) = p(x, n-1)x + a_n$

Horner scheme – Proof

Theorem. Value of a polynomial

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n$$

can be computed using the following recursive function:

$p(x,0) = a_0$

$p(x,n) = p(x, n-1)x + a_n$

Proof:

$$p(x,0) = a_0x^n + \cancel{a_1x^{n-1} + .. + a_{n-1}x^1 + a_n} = a_0\boxed{x^0} = a_0$$

Horner scheme – Proof

Theorem. Value of a polynomial

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n$$

can be computed using the following recursive function:

$p(x,0) = a_0$

$p(x,n) = p(x, n-1)x + a_n$

Proof:

$$p(x,0) = a_0x^n + \cancel{a_1x^{n-1} + .. + a_{n-1}x^1 + a_n} = a_0\,x^0 = a_0$$

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n =$$

$$= (a_0x^{n-1} + a_1x^{n-2} + .. + a_{n-1})x + a_n = p(x, n-1)x + a_n$$

Horner scheme

Theorem. Value of a polynomial

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n$$

can be computed using the following recursive function:

$p(x,0) = a_0$

$p(x,n) = p(x, n-1)x + a_n$

C

```c
double p(double x, int n, double a[ ]){
        if (n == 0) return a[0];
        else return p(x, n-1, a)*x + a[n];
        }
```


Horner scheme

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n$$

$p(x,8) = 10^{300}\,x^8$

$p(10^{-50},8) = 10^{-100}$

```c
#include <stdio.h>
double p(double x, int n, double a[ ]){
        if (n == 0) return a[0];
        else return p(x, n-1, a)*x + a[n];
        }
  void main(){
        double a[]= {1e300, 0, 0, 0, 0, 0, 0, 0, 0};
        printf("%g\n", p(1e-50, 8, a));
        return;
        }
```

$p(10^{-50},0) = 10^{300}$

$p(10^{-50},1) = 10^{250}$

$p(10^{-50},2) = 10^{200}$

$p(10^{-50},3) = 10^{150}$

$p(10^{-50},4) = 10^{100}$

$p(10^{-50},5) = 10^{50}$

$p(10^{-50},6) = 10^{0}$

$p(10^{-50},7) = 10^{-50}$

$p(10^{-50},8) = 10^{-100}$

```
1e-100
```


Horner scheme

python

$$p(x,n) = a_0x^n + a_1x^{n-1} + .. + a_{n-1}x^1 + a_n$$

$p(x,8) = 10^{300}\,x^8$

$p(10^{-50},8) = 10^{-100}$

```python
def p(x, n, a):
    if n==0:
        return a[0]
    else:
        return p(x, n-1, a)*x + a[n]
a=[1e300, 0, 0, 0, 0, 0, 0, 0, 0]
print p(1e-50, 8, a)
```

$p(10^{-50},0) = 10^{300}$

$p(10^{-50},1) = 10^{250}$

$p(10^{-50},2) = 10^{200}$

$p(10^{-50},3) = 10^{150}$

$p(10^{-50},4) = 10^{100}$

$p(10^{-50},5) = 10^{50}$

$p(10^{-50},6) = 10^{0}$

$p(10^{-50},7) = 10^{-50}$

$p(10^{-50},8) = 10^{-100}$

```
1e-100
```


Ill-conditioned problems

A problem is ill-conditioned if:
A small relative error in the data → much larger relative error in the result(s)

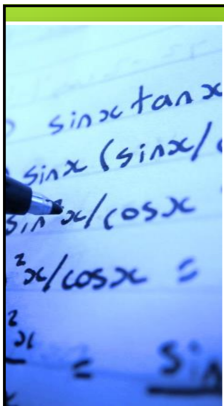
$$p(x) = a_{20}\,x^{20} + \ldots + a_1\,x + a_0 = \prod_{k=1}^{20}(x-k) = 0$$

$x = 1, 2, \ldots, 20$

$a_{19} = -210 \rightarrow a_{19} = -(210 + 2^{-23})$

$x = 15 \rightarrow x = 13{,}99 + 2{,}5i$

Agenda



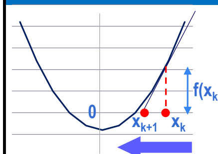
Square root

$g(a) = \sqrt{a}$

Transformation to finding zeroes of f(x)

$f(x) = x^2 - a = 0$

Newton method:



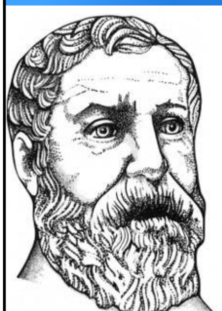
Geom. interpretation of derivative:

$f'(x_k) = f(x_k) / (x_k - x_{k+1})$

where $f'(x) = 2x$. After rewriting:

$x_{k+1} = ½ (x_k + a/x_k)$

Iterative algorithm



Heron of Alexandria
From a German translation of *Pneumatica*, 1688 r.
http://pl.wikipedia.org/wiki/Heron_z_Aleksandrii

Heron's algorithm of computing a square root of *a*

$$x_1 = \begin{cases} a & \text{if } a >= 1 \\ 1 & \text{if } a < 1 \end{cases}$$

$x_{k+1} = ½ (x_k + a/x_k)$

Conditional expression

C

```
cond ? val1 : val2
a >= 1 ?   a  :   1
```

```
val1 if cond   else val2
  a  if a >= 1 else  1
```


Heron's method

Heron's algorithm of computing a square root of *a*

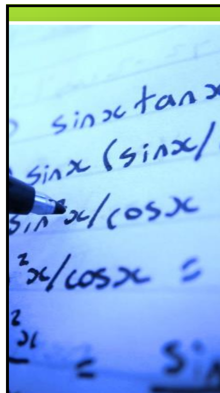
$$x_1 = \begin{cases} a & \text{if } a >= 1 \\ 1 & \text{if } a < 1 \end{cases}$$

$x_{k+1} = ½ (x_k + a/x_k)$

```
MaxErr= 0.03
def SqR(a):
    X= a if a >= 1 else 1
    NewX= 0.5*(X + a/X)
    Err= (NewX - X)/NewX
    if Err < 0: Err= -Err
    while Err > MaxErr:
          X= NewX
          NewX= 0.5*(X + a/X)
          Err= (NewX - X)/NewX
          if Err < 0: Err= -Err
    return NewX
print("Square root of 2 = ", SqR(2))
```

```
2:   1.41421568627
```


Agenda



Maclaurin series

$$f(x) \cong \sum_{k=0}^{N} \frac{\boxed{f^{(k)}(0)}\, x^k}{k!}$$

f(x) = e^x

- $(e^x)' = e^x$
- $e^0 = 1$

$$e^x \approx \sum_{k=0}^{N} x^k / k!$$

$e^x \cong x^0/0! + x^1/1! + x^2/2! + x^3/3! + ..$

$= 1 + x / 1! + x^2/2! + x^3/3! + .$

e^x

$e^x = 1 + x/1! + x^2/2! + x^3/3! + x^4/4! + ...$

e(1) = 2,71..
e(0) = 1

$T = \frac{num}{den}$ $\quad \frac{1}{1} \quad \frac{x}{1!} \quad \frac{x^2}{2!} \quad \frac{x^3}{3!}$

e^x

$e^x = 1 + x/1! + x^2/2! + x^3/3! + x^4/4! + ...$

e(1) = 2,71..
e(0) = 1

$T = \frac{num}{den}$

$$\frac{1}{1} \xrightarrow[*1]{*x} \frac{x}{1!} \xrightarrow[*2]{*x} \frac{x^2}{2!} \xrightarrow[*3]{*x} \frac{x^3}{3!}$$

e^x

$$\frac{1}{1} \xrightarrow[*1]{*x} \frac{x}{1!} \xrightarrow[*2]{*x} \frac{x^2}{2!} \xrightarrow[*3]{*x} \frac{x^3}{3!}$$

C

```c
#include <stdio.h>
#define N 7
double e(double x){
        double Sum=0, T;
        double num= 1.0;
        double den=1.0;
        int i;
        for (i=1; i<=N; i++){
            T= num/den;
            Sum+= T;
            num*= x;
            den*= i;
            }
        return Sum;}
int main (){
    printf("%g \n", e(1));}
```

2.71806

e^x

$$\frac{1}{1} \xrightarrow[*1]{*x} \frac{x}{1!} \xrightarrow[*2]{*x} \frac{x^2}{2!} \xrightarrow[*3]{*x} \frac{x^3}{3!}$$

```python
N= 7
def e(x):
    Sum= 0.0
    num= 1.0
    den= 1.0
    for i in range(1, N+1):
        Term= num/den
        Sum+= Term
        num*= x
        den*= i
    return Sum
print(e(1))
```

2.7180555555555554

Maclaurin series

$$f(x) \cong \sum_{k=0}^{N} \frac{f^{(k)}(0)\, x^k}{k!}$$

f(x) = cos x

- $(\cos x)' = -\sin x$
- $(\cos x)'' = -\cos x$
- $(\cos x)^{(3)} = \sin x$
- $(\cos x)^{(4)} = \cos x$
- . . .

Maclaurin series

$$f(x) \cong \sum_{k=0}^{N} \frac{\boxed{f^{(k)}(0)} x^k}{k!}$$

f(x) = cos x

(cos x)$^{(0)}$ = cos x	1
(cos x)' = – sin x	0
(cos x)'' = – cos x	-1
(cos x)$^{(3)}$ = sin x	0
(cos x)$^{(4)}$ = cos x	1
. . .	

cos x ≈ 1 + 0 – x^2/2! + 0 + x^4/4! + ..

cos(x)

cos x ≈ 1 + 0 – x^2/2! + 0 + x^4/4! + ..

cos(0) = 1
cos(1,57..) = 0

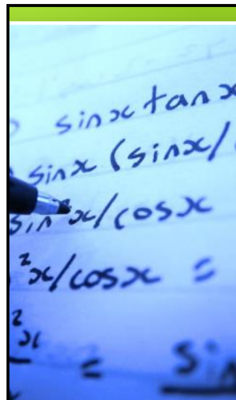
$$T = \frac{num}{den} \qquad \frac{1}{1} \qquad \frac{-x^2}{2!} \qquad \frac{x^4}{4!}$$

cos(x)

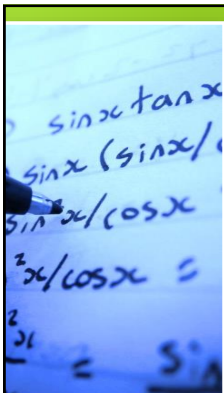
cos x ≈ 1 + 0 – x^2/2! + 0 + x^4/4! + ..

cos(0) = 1
cos(1,57..) = 0

$$T = \frac{num}{den} \qquad \frac{1}{1} \xrightarrow[*1*2]{*(-x^2)} \frac{-x^2}{2!} \xrightarrow[*3*4]{*(-x^2)} \frac{x^4}{4!}$$



Agenda



Next lecture

No.	Topic	Date
1	Imperative Programming	2023-10-09
2	Digital Circuits	2023-10-16
3	Computers	2023-10-23
4	Subprograms	2023-11-06
5	Text Processing	2023-11-13
6	Object-oriented Programming	2023-11-20
7	Numerical methods	2023-11-27
8	Computational Complexity	2023-12-04
9	Databases and Machine Learning	2023-12-11
10	Parallel Processing	2023-12-18
11	Computer Networks & Cybersecurit	2024-01-08
12	Software Engineering	2024-01-15
13	Embedded Systems	2024-01-22
14	Professionalism in Computing	2024-01-29