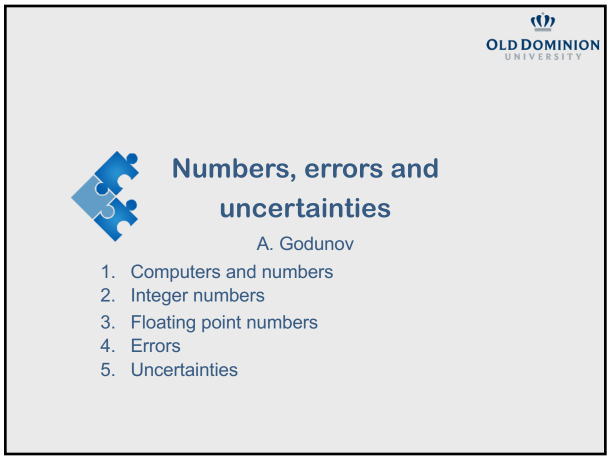
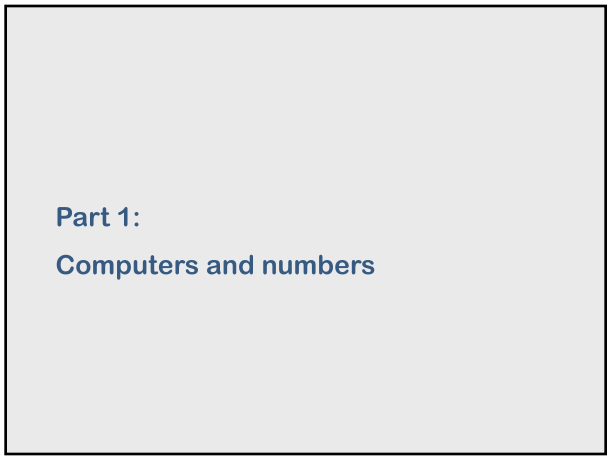

Numbers, errors and uncertainties

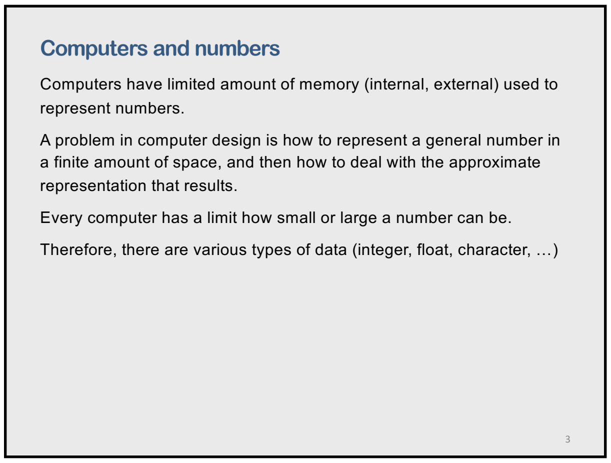
A. Godunov

1


Part 1:

Computers and numbers

2


Computers and numbers

Computers have limited amount of memory (internal, external) used to represent numbers.

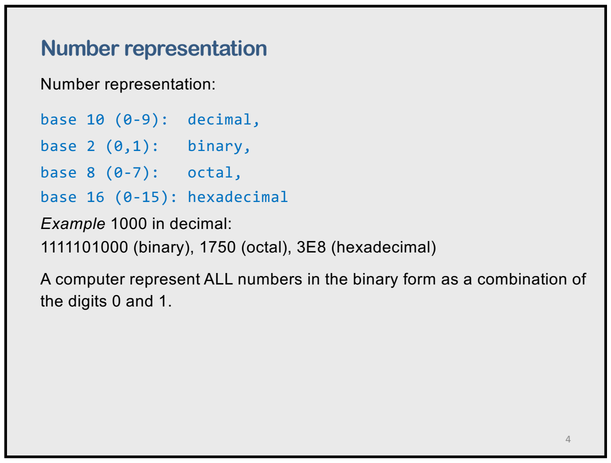
A problem in computer design is how to represent a general number in a finite amount of space, and then how to deal with the approximate representation that results.

Every computer has a limit how small or large a number can be.

Therefore, there are various types of data (integer, float, character, …)

3

3


Number representation

Number representation:

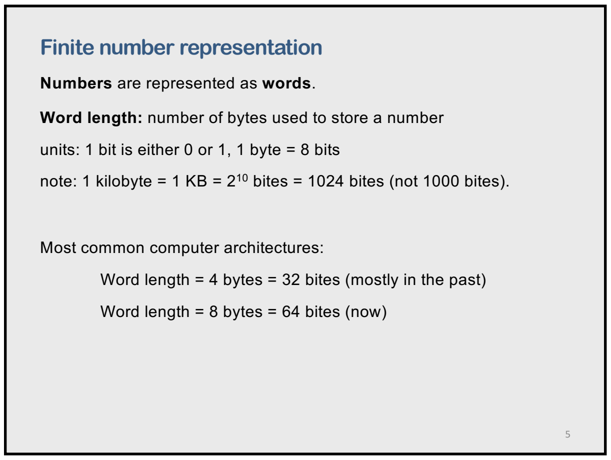
```
base 10 (0-9):  decimal,
base 2 (0,1):   binary,
base 8 (0-7):   octal,
base 16 (0-15): hexadecimal
```

Example 1000 in decimal:
1111101000 (binary), 1750 (octal), 3E8 (hexadecimal)

A computer represent ALL numbers in the binary form as a combination of the digits 0 and 1.

4

4


Finite number representation

Numbers are represented as **words**.

Word length: number of bytes used to store a number

units: 1 bit is either 0 or 1, 1 byte = 8 bits

note: 1 kilobyte = 1 KB = 2^{10} bites = 1024 bites (not 1000 bites).

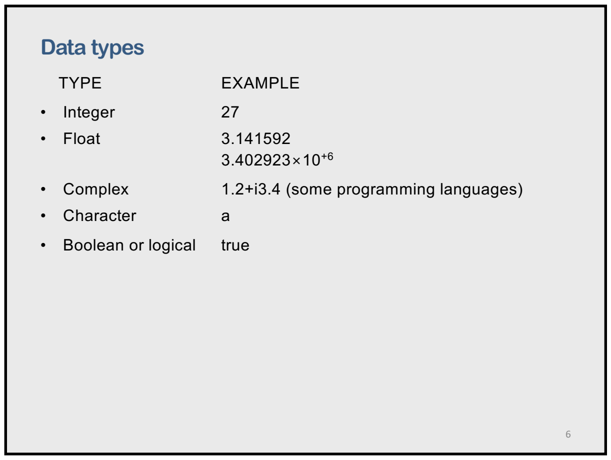
Most common computer architectures:

Word length = 4 bytes = 32 bites (mostly in the past)

Word length = 8 bytes = 64 bites (now)

5

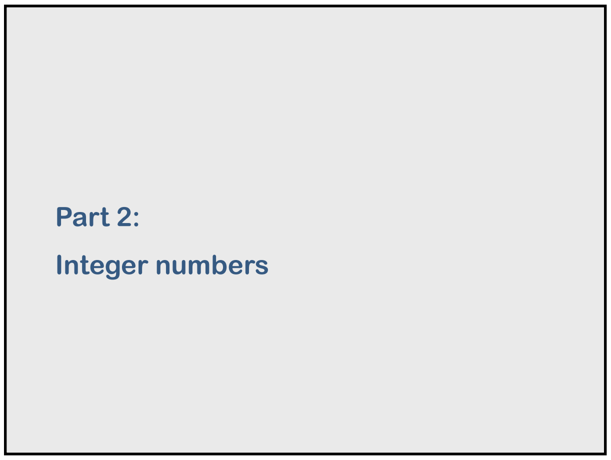
5


Data types

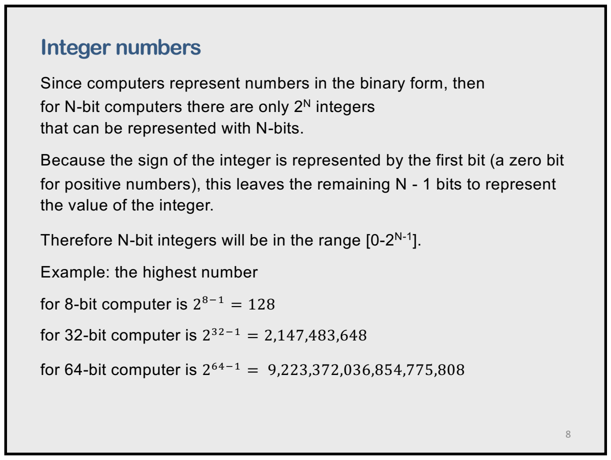
TYPE	EXAMPLE
• Integer	27
• Float	3.141592 $3.402923\times10^{+6}$
• Complex	1.2+i3.4 (some programming languages)
• Character	a
• Boolean or logical	true

6

6


Part 2:
Integer numbers

7


Integer numbers

Since computers represent numbers in the binary form, then for N-bit computers there are only 2^N integers that can be represented with N-bits.

Because the sign of the integer is represented by the first bit (a zero bit for positive numbers), this leaves the remaining N - 1 bits to represent the value of the integer.

Therefore N-bit integers will be in the range [0-2^{N-1}].

Example: the highest number

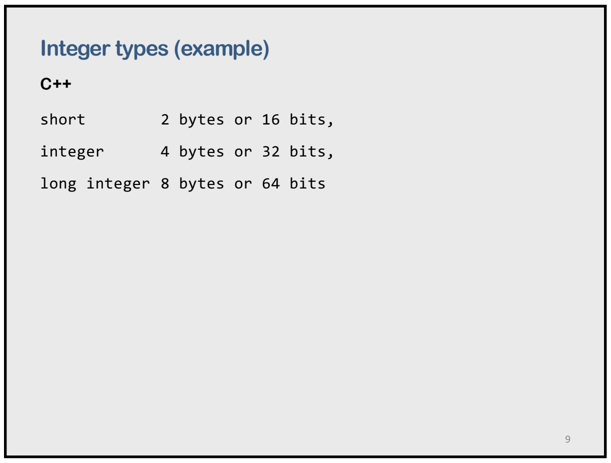
for 8-bit computer is $2^{8-1} = 128$

for 32-bit computer is $2^{32-1} = 2{,}147{,}483{,}648$

for 64-bit computer is $2^{64-1} = 9{,}223{,}372{,}036{,}854{,}775{,}808$

8

8

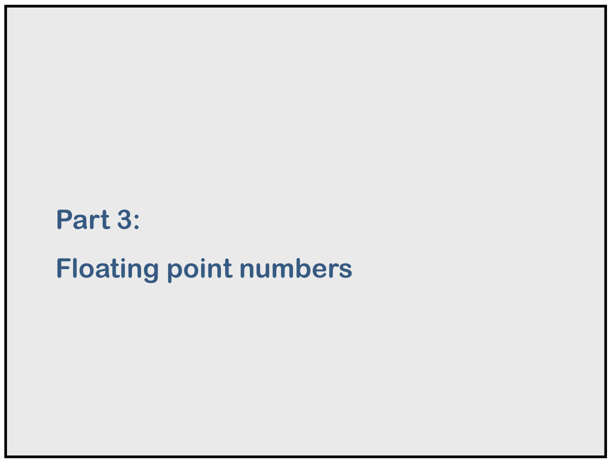

Integer types (example)

C++

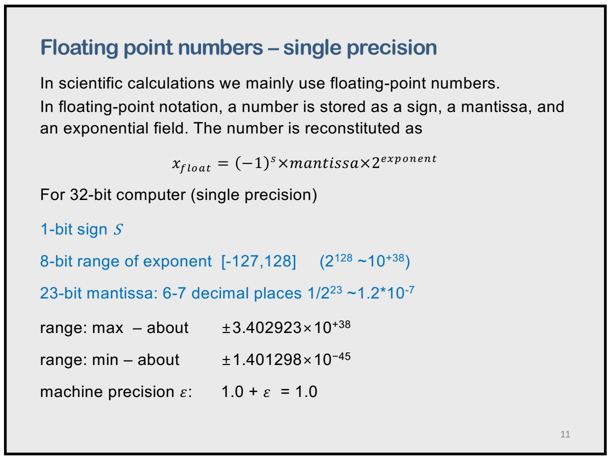
```
short        2 bytes or 16 bits,
integer      4 bytes or 32 bits,
long integer 8 bytes or 64 bits
```

9

9


Part 3:
Floating point numbers

10


Floating point numbers – single precision

In scientific calculations we mainly use floating-point numbers.
In floating-point notation, a number is stored as a sign, a mantissa, and an exponential field. The number is reconstituted as

$$x_{float} = (-1)^s \times mantissa \times 2^{exponent}$$

For 32-bit computer (single precision)

1-bit sign S

8-bit range of exponent [-127,128] (2^{128} ~10^{+38})

23-bit mantissa: 6-7 decimal places $1/2^{23}$ ~$1.2*10^{-7}$

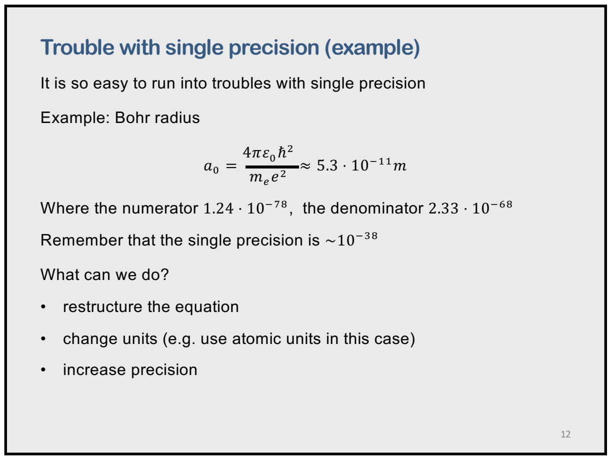
range: max – about ±$3.402923\times10^{+38}$

range: min – about ±1.401298×10^{-45}

machine precision ε: $1.0 + \varepsilon = 1.0$

11

11


Trouble with single precision (example)

It is so easy to run into troubles with single precision

Example: Bohr radius

$$a_0 = \frac{4\pi\varepsilon_0\hbar^2}{m_e e^2} \approx 5.3 \cdot 10^{-11} m$$

Where the numerator $1.24 \cdot 10^{-78}$, the denominator $2.33 \cdot 10^{-68}$

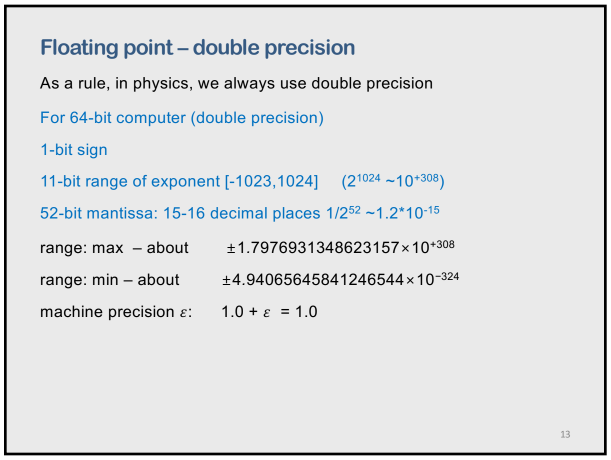
Remember that the single precision is ~10^{-38}

What can we do?

- restructure the equation
- change units (e.g. use atomic units in this case)
- increase precision

12

12



Floating point – double precision

As a rule, in physics, we always use double precision

For 64-bit computer (double precision)

1-bit sign

11-bit range of exponent [-1023,1024] $(2^{1024} \sim 10^{+308})$

52-bit mantissa: 15-16 decimal places $1/2^{52} \sim 1.2*10^{-15}$

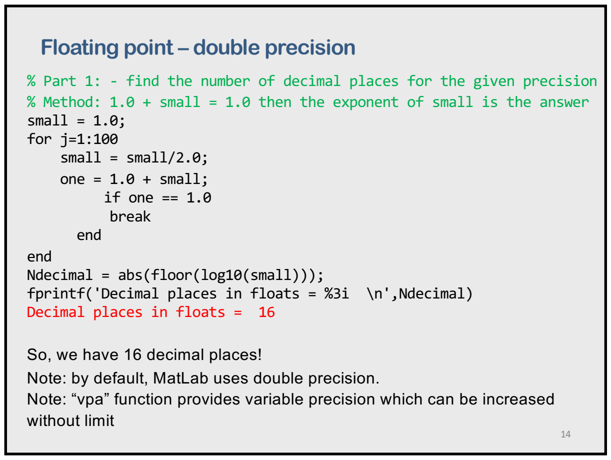
range: max – about $\pm 1.7976931348623157 \times 10^{+308}$

range: min – about $\pm 4.940656458412465 44 \times 10^{-324}$

machine precision ε: $1.0 + \varepsilon = 1.0$

13

13



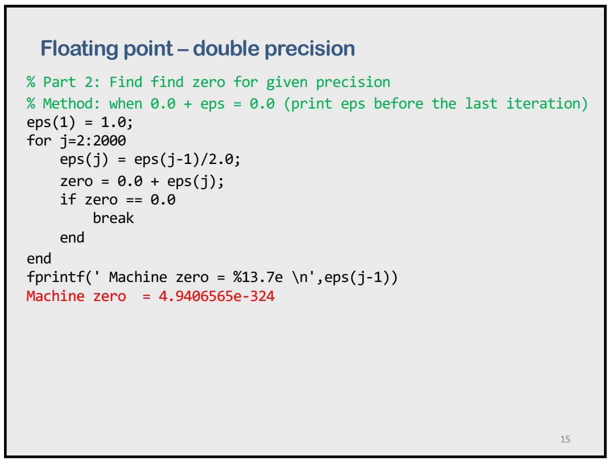
Floating point – double precision

```
% Part 1: - find the number of decimal places for the given precision
% Method: 1.0 + small = 1.0 then the exponent of small is the answer
small = 1.0;
for j=1:100
    small = small/2.0;
    one = 1.0 + small;
         if one == 1.0
          break
      end
end
Ndecimal = abs(floor(log10(small)));
fprintf('Decimal places in floats = %3i  \n',Ndecimal)
Decimal places in floats =  16
```

So, we have 16 decimal places!
Note: by default, MatLab uses double precision.
Note: "vpa" function provides variable precision which can be increased without limit

14

14

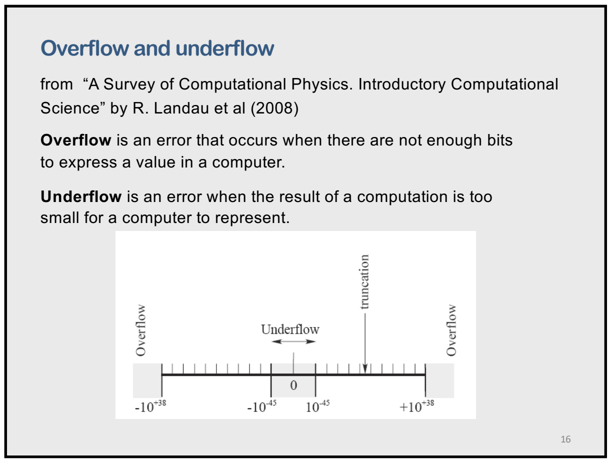


Floating point – double precision

```
% Part 2: Find find zero for given precision
% Method: when 0.0 + eps = 0.0 (print eps before the last iteration)
eps(1) = 1.0;
for j=2:2000
    eps(j) = eps(j-1)/2.0;
    zero = 0.0 + eps(j);
    if zero == 0.0
        break
    end
end
fprintf(' Machine zero = %13.7e \n',eps(j-1))
Machine zero  = 4.9406565e-324
```

15

15

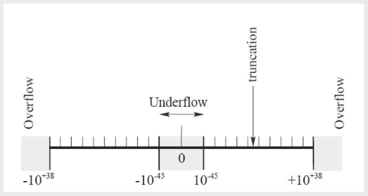


Overflow and underflow

from "A Survey of Computational Physics. Introductory Computational Science" by R. Landau et al (2008)

Overflow is an error that occurs when there are not enough bits to express a value in a computer.

Underflow is an error when the result of a computation is too small for a computer to represent.



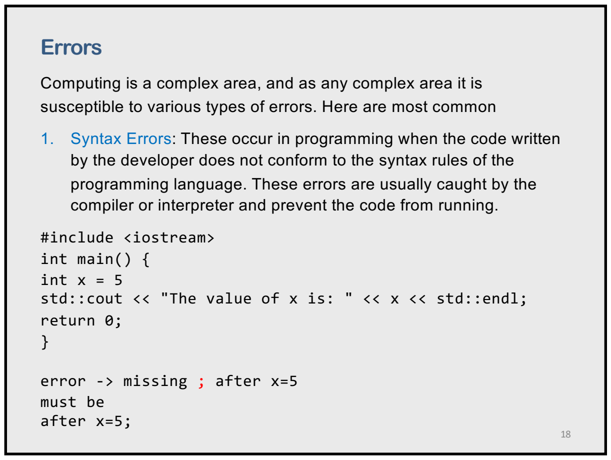
16

16

Part 4:

Errors

17



Errors

Computing is a complex area, and as any complex area it is susceptible to various types of errors. Here are most common

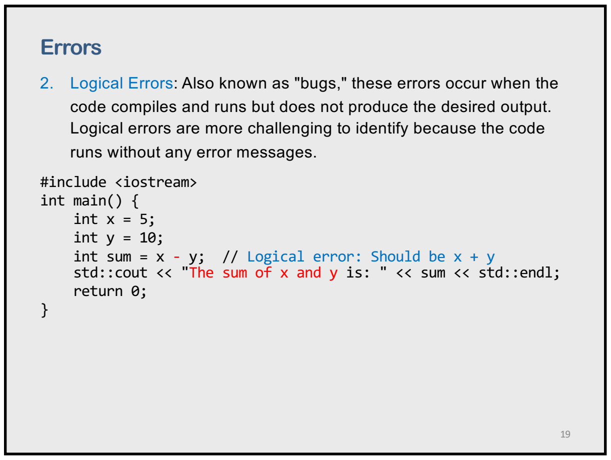
1. Syntax Errors: These occur in programming when the code written by the developer does not conform to the syntax rules of the programming language. These errors are usually caught by the compiler or interpreter and prevent the code from running.

```
#include <iostream>
int main() {
int x = 5
std::cout << "The value of x is: " << x << std::endl;
return 0;
}

error -> missing ; after x=5
must be
after x=5;
```

18

18

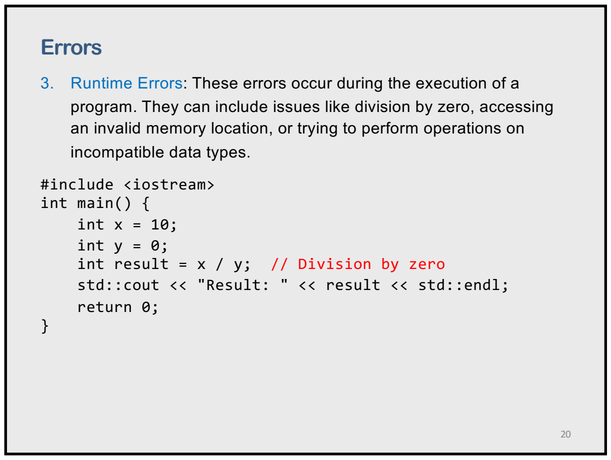


Errors

2. Logical Errors: Also known as "bugs," these errors occur when the code compiles and runs but does not produce the desired output. Logical errors are more challenging to identify because the code runs without any error messages.

```
#include <iostream>
int main() {
    int x = 5;
    int y = 10;
    int sum = x - y;  // Logical error: Should be x + y
    std::cout << "The sum of x and y is: " << sum << std::endl;
    return 0;
}
```

19

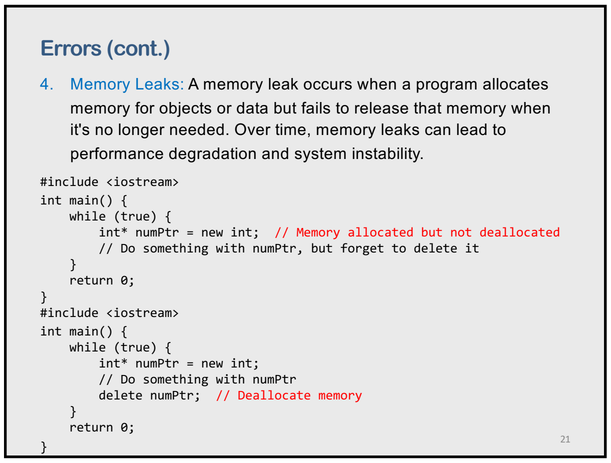


Errors

3. Runtime Errors: These errors occur during the execution of a program. They can include issues like division by zero, accessing an invalid memory location, or trying to perform operations on incompatible data types.

```
#include <iostream>
int main() {
    int x = 10;
    int y = 0;
    int result = x / y;  // Division by zero
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

20

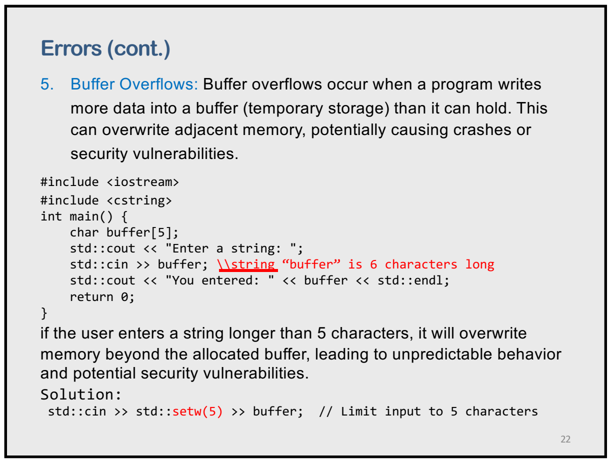


Errors (cont.)

4. Memory Leaks: A memory leak occurs when a program allocates memory for objects or data but fails to release that memory when it's no longer needed. Over time, memory leaks can lead to performance degradation and system instability.

```
#include <iostream>
int main() {
    while (true) {
        int* numPtr = new int;  // Memory allocated but not deallocated
        // Do something with numPtr, but forget to delete it
    }
    return 0;
}
#include <iostream>
int main() {
    while (true) {
        int* numPtr = new int;
        // Do something with numPtr
        delete numPtr;  // Deallocate memory
    }
    return 0;
}
```

21



Errors (cont.)

5. Buffer Overflows: Buffer overflows occur when a program writes more data into a buffer (temporary storage) than it can hold. This can overwrite adjacent memory, potentially causing crashes or security vulnerabilities.

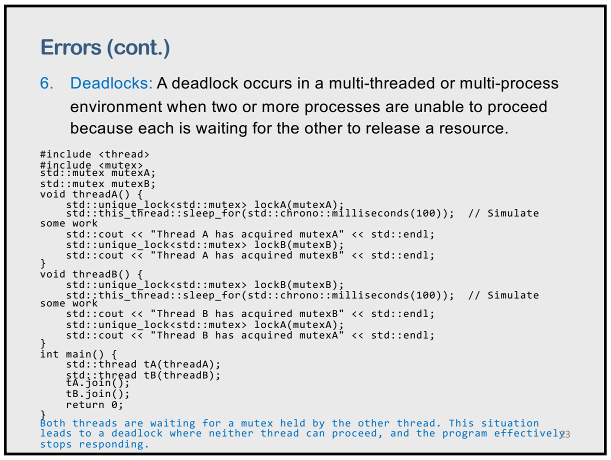
```
#include <iostream>
#include <cstring>
int main() {
    char buffer[5];
    std::cout << "Enter a string: ";
    std::cin >> buffer; \\string “buffer” is 6 characters long
    std::cout << "You entered: " << buffer << std::endl;
    return 0;
}
```

if the user enters a string longer than 5 characters, it will overwrite memory beyond the allocated buffer, leading to unpredictable behavior and potential security vulnerabilities.

Solution:

```
 std::cin >> std::setw(5) >> buffer;  // Limit input to 5 characters
```

22



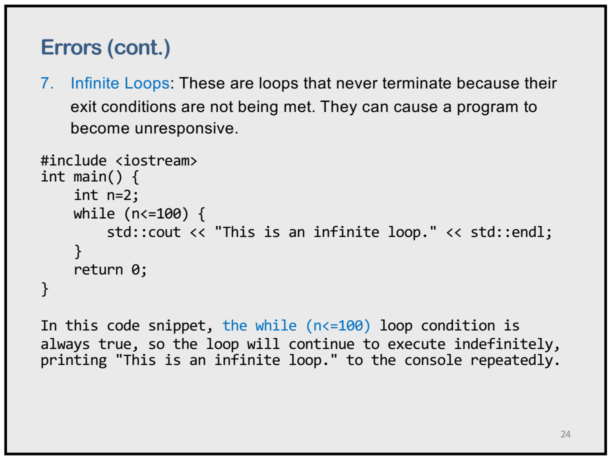
Errors (cont.)

6. Deadlocks: A deadlock occurs in a multi-threaded or multi-process environment when two or more processes are unable to proceed because each is waiting for the other to release a resource.

```
#include <thread>
#include <mutex>
std::mutex mutexA;
std::mutex mutexB;
void threadA() {
    std::unique_lock<std::mutex> lockA(mutexA);
    std::this_thread::sleep_for(std::chrono::milliseconds(100));  // Simulate
some work
    std::cout << "Thread A has acquired mutexA" << std::endl;
    std::unique_lock<std::mutex> lockB(mutexB);
    std::cout << "Thread A has acquired mutexB" << std::endl;
}
void threadB() {
    std::unique_lock<std::mutex> lockB(mutexB);
    std::this_thread::sleep_for(std::chrono::milliseconds(100));  // Simulate
some work
    std::cout << "Thread B has acquired mutexB" << std::endl;
    std::unique_lock<std::mutex> lockA(mutexA);
    std::cout << "Thread B has acquired mutexA" << std::endl;
}
int main() {
    std::thread tA(threadA);
    std::thread tB(threadB);
    tA.join();
    tB.join();
    return 0;
}
```

Both threads are waiting for a mutex held by the other thread. This situation leads to a deadlock where neither thread can proceed, and the program effectively stops responding.

23



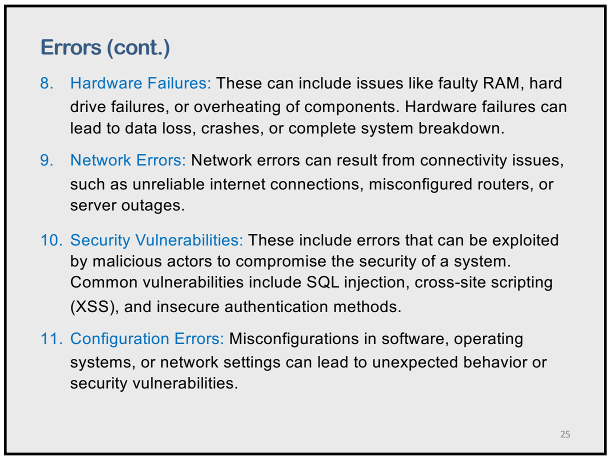
Errors (cont.)

7. Infinite Loops: These are loops that never terminate because their exit conditions are not being met. They can cause a program to become unresponsive.

```
#include <iostream>
int main() {
    int n=2;
    while (n<=100) {
        std::cout << "This is an infinite loop." << std::endl;
    }
    return 0;
}
```

In this code snippet, the while (n<=100) loop condition is always true, so the loop will continue to execute indefinitely, printing "This is an infinite loop." to the console repeatedly.

24

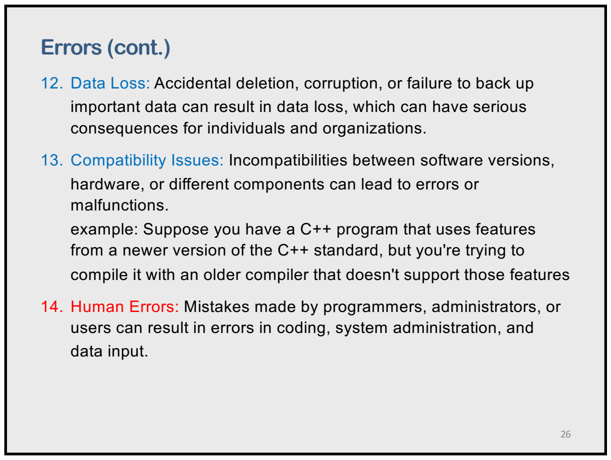


Errors (cont.)

8. Hardware Failures: These can include issues like faulty RAM, hard drive failures, or overheating of components. Hardware failures can lead to data loss, crashes, or complete system breakdown.
9. Network Errors: Network errors can result from connectivity issues, such as unreliable internet connections, misconfigured routers, or server outages.
10. Security Vulnerabilities: These include errors that can be exploited by malicious actors to compromise the security of a system. Common vulnerabilities include SQL injection, cross-site scripting (XSS), and insecure authentication methods.
11. Configuration Errors: Misconfigurations in software, operating systems, or network settings can lead to unexpected behavior or security vulnerabilities.

25

25

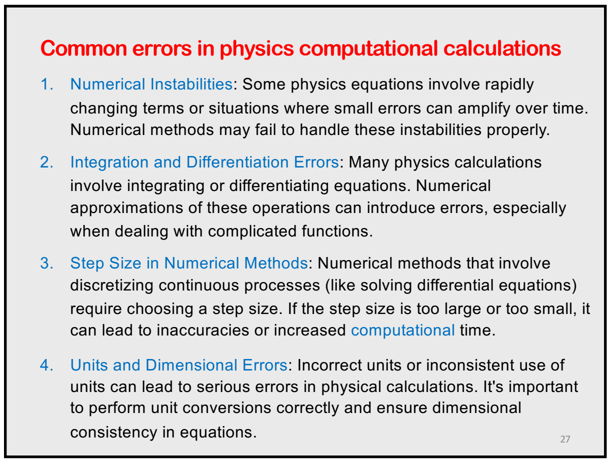


Errors (cont.)

12. Data Loss: Accidental deletion, corruption, or failure to back up important data can result in data loss, which can have serious consequences for individuals and organizations.
13. Compatibility Issues: Incompatibilities between software versions, hardware, or different components can lead to errors or malfunctions.
    example: Suppose you have a C++ program that uses features from a newer version of the C++ standard, but you're trying to compile it with an older compiler that doesn't support those features
14. Human Errors: Mistakes made by programmers, administrators, or users can result in errors in coding, system administration, and data input.

26

26

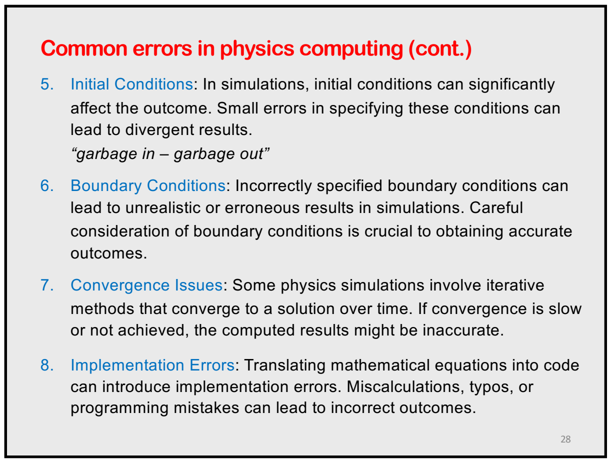


Common errors in physics computational calculations

1. Numerical Instabilities: Some physics equations involve rapidly changing terms or situations where small errors can amplify over time. Numerical methods may fail to handle these instabilities properly.
2. Integration and Differentiation Errors: Many physics calculations involve integrating or differentiating equations. Numerical approximations of these operations can introduce errors, especially when dealing with complicated functions.
3. Step Size in Numerical Methods: Numerical methods that involve discretizing continuous processes (like solving differential equations) require choosing a step size. If the step size is too large or too small, it can lead to inaccuracies or increased computational time.
4. Units and Dimensional Errors: Incorrect units or inconsistent use of units can lead to serious errors in physical calculations. It's important to perform unit conversions correctly and ensure dimensional consistency in equations.

27

27

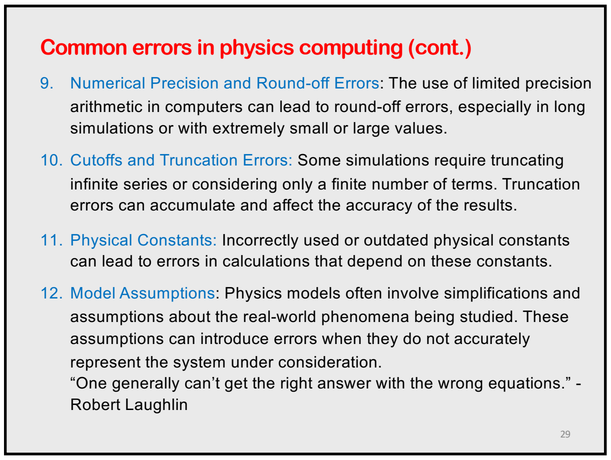


Common errors in physics computing (cont.)

5. Initial Conditions: In simulations, initial conditions can significantly affect the outcome. Small errors in specifying these conditions can lead to divergent results.
   "garbage in – garbage out"
6. Boundary Conditions: Incorrectly specified boundary conditions can lead to unrealistic or erroneous results in simulations. Careful consideration of boundary conditions is crucial to obtaining accurate outcomes.
7. Convergence Issues: Some physics simulations involve iterative methods that converge to a solution over time. If convergence is slow or not achieved, the computed results might be inaccurate.
8. Implementation Errors: Translating mathematical equations into code can introduce implementation errors. Miscalculations, typos, or programming mistakes can lead to incorrect outcomes.

28

28

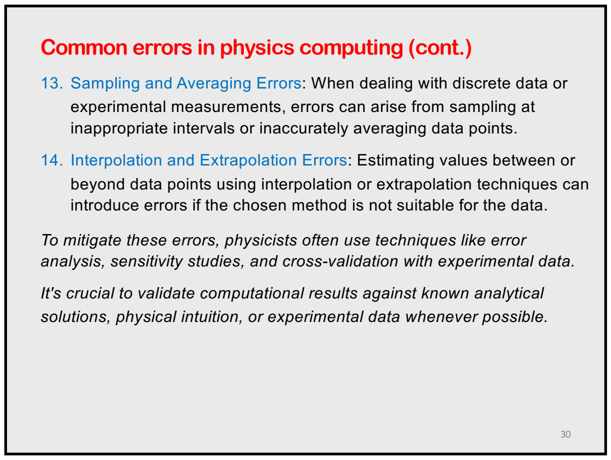


Common errors in physics computing (cont.)

9. Numerical Precision and Round-off Errors: The use of limited precision arithmetic in computers can lead to round-off errors, especially in long simulations or with extremely small or large values.
10. Cutoffs and Truncation Errors: Some simulations require truncating infinite series or considering only a finite number of terms. Truncation errors can accumulate and affect the accuracy of the results.
11. Physical Constants: Incorrectly used or outdated physical constants can lead to errors in calculations that depend on these constants.
12. Model Assumptions: Physics models often involve simplifications and assumptions about the real-world phenomena being studied. These assumptions can introduce errors when they do not accurately represent the system under consideration.
    "One generally can't get the right answer with the wrong equations." - Robert Laughlin

29

29



Common errors in physics computing (cont.)

13. Sampling and Averaging Errors: When dealing with discrete data or experimental measurements, errors can arise from sampling at inappropriate intervals or inaccurately averaging data points.
14. Interpolation and Extrapolation Errors: Estimating values between or beyond data points using interpolation or extrapolation techniques can introduce errors if the chosen method is not suitable for the data.

To mitigate these errors, physicists often use techniques like error analysis, sensitivity studies, and cross-validation with experimental data.

It's crucial to validate computational results against known analytical solutions, physical intuition, or experimental data whenever possible.

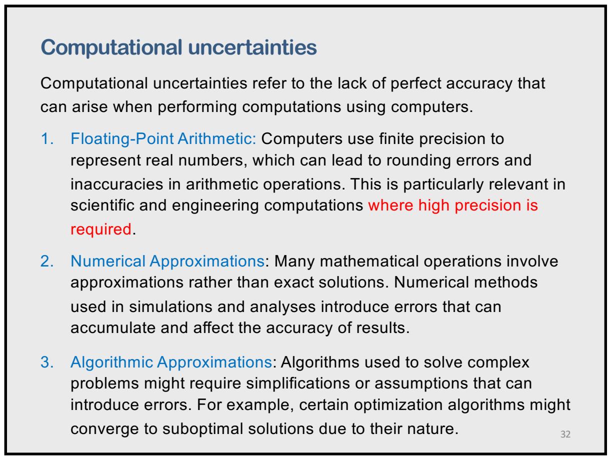
30

30

Part 5:

Uncertainties

31

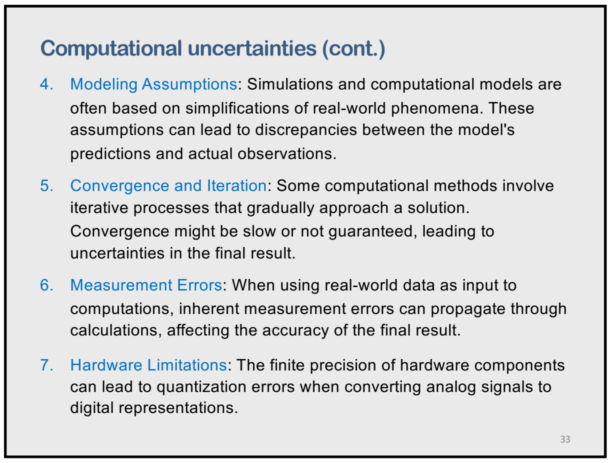


Computational uncertainties

Computational uncertainties refer to the lack of perfect accuracy that can arise when performing computations using computers.

1. Floating-Point Arithmetic: Computers use finite precision to represent real numbers, which can lead to rounding errors and inaccuracies in arithmetic operations. This is particularly relevant in scientific and engineering computations where high precision is required.
2. Numerical Approximations: Many mathematical operations involve approximations rather than exact solutions. Numerical methods used in simulations and analyses introduce errors that can accumulate and affect the accuracy of results.
3. Algorithmic Approximations: Algorithms used to solve complex problems might require simplifications or assumptions that can introduce errors. For example, certain optimization algorithms might converge to suboptimal solutions due to their nature.

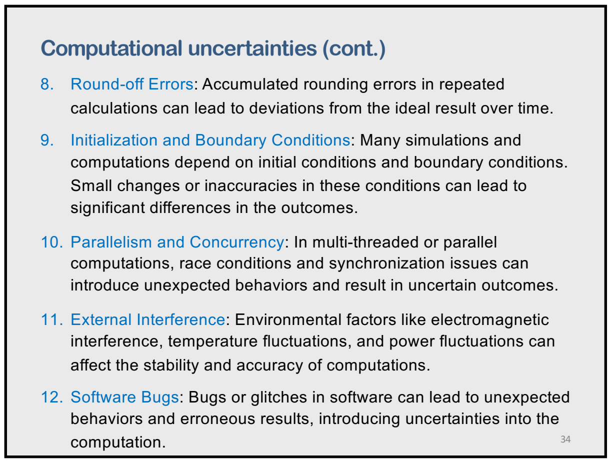
32



Computational uncertainties (cont.)

4. Modeling Assumptions: Simulations and computational models are often based on simplifications of real-world phenomena. These assumptions can lead to discrepancies between the model's predictions and actual observations.
5. Convergence and Iteration: Some computational methods involve iterative processes that gradually approach a solution. Convergence might be slow or not guaranteed, leading to uncertainties in the final result.
6. Measurement Errors: When using real-world data as input to computations, inherent measurement errors can propagate through calculations, affecting the accuracy of the final result.
7. Hardware Limitations: The finite precision of hardware components can lead to quantization errors when converting analog signals to digital representations.

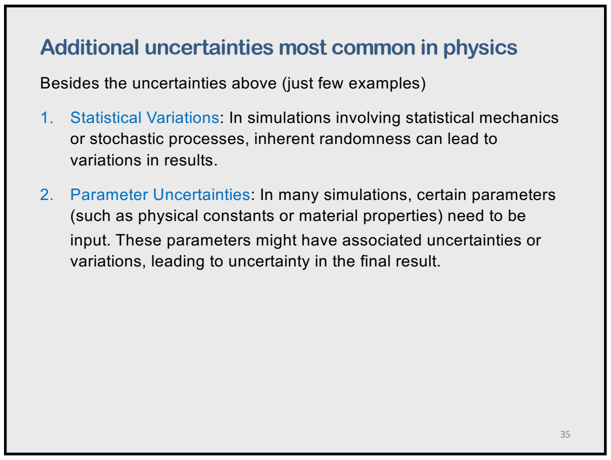
33



Computational uncertainties (cont.)

8. Round-off Errors: Accumulated rounding errors in repeated calculations can lead to deviations from the ideal result over time.
9. Initialization and Boundary Conditions: Many simulations and computations depend on initial conditions and boundary conditions. Small changes or inaccuracies in these conditions can lead to significant differences in the outcomes.
10. Parallelism and Concurrency: In multi-threaded or parallel computations, race conditions and synchronization issues can introduce unexpected behaviors and result in uncertain outcomes.
11. External Interference: Environmental factors like electromagnetic interference, temperature fluctuations, and power fluctuations can affect the stability and accuracy of computations.
12. Software Bugs: Bugs or glitches in software can lead to unexpected behaviors and erroneous results, introducing uncertainties into the computation.

34



Additional uncertainties most common in physics

Besides the uncertainties above (just few examples)

1. Statistical Variations: In simulations involving statistical mechanics or stochastic processes, inherent randomness can lead to variations in results.
2. Parameter Uncertainties: In many simulations, certain parameters (such as physical constants or material properties) need to be input. These parameters might have associated uncertainties or variations, leading to uncertainty in the final result.

35