CS 101
Computer Programming and Utilization

Basics

Suyash P. Awate

Computer

- "A machine that can be *programmed* to carry out sequences of arithmetic or logical operations (computation) automatically."
  - en.wikipedia.org/wiki/Computer

PROGRAM = योजना [pr. {yojana}] (Verb)

Usage : The computer is **programm**ed to receive instructions from the user.
उदाहरण : योजना अपवाद उत्पन्न हुआ

+23

PROGRAM = कार्यक्रम [pr. {karyakram}] (Noun)

Usage : he proposed an elaborate **program** of public works
उदाहरण : क्या कर्ता कार्यक्रम के प्रति प्रतिक्रिया दे रहा है

program

verb [T]

UK /ˈprəʊ.græm/ US /ˈproʊ.
-mm-

to write a series of instructions that make a computer perform a particular operation:

- [+ to infinitive] *She programmed the computer **to** calculate the rate of exchange in twelve currencies.*

Cambridge Dictionary

program

noun [C]

US /ˈproʊ·græm, -grəm/

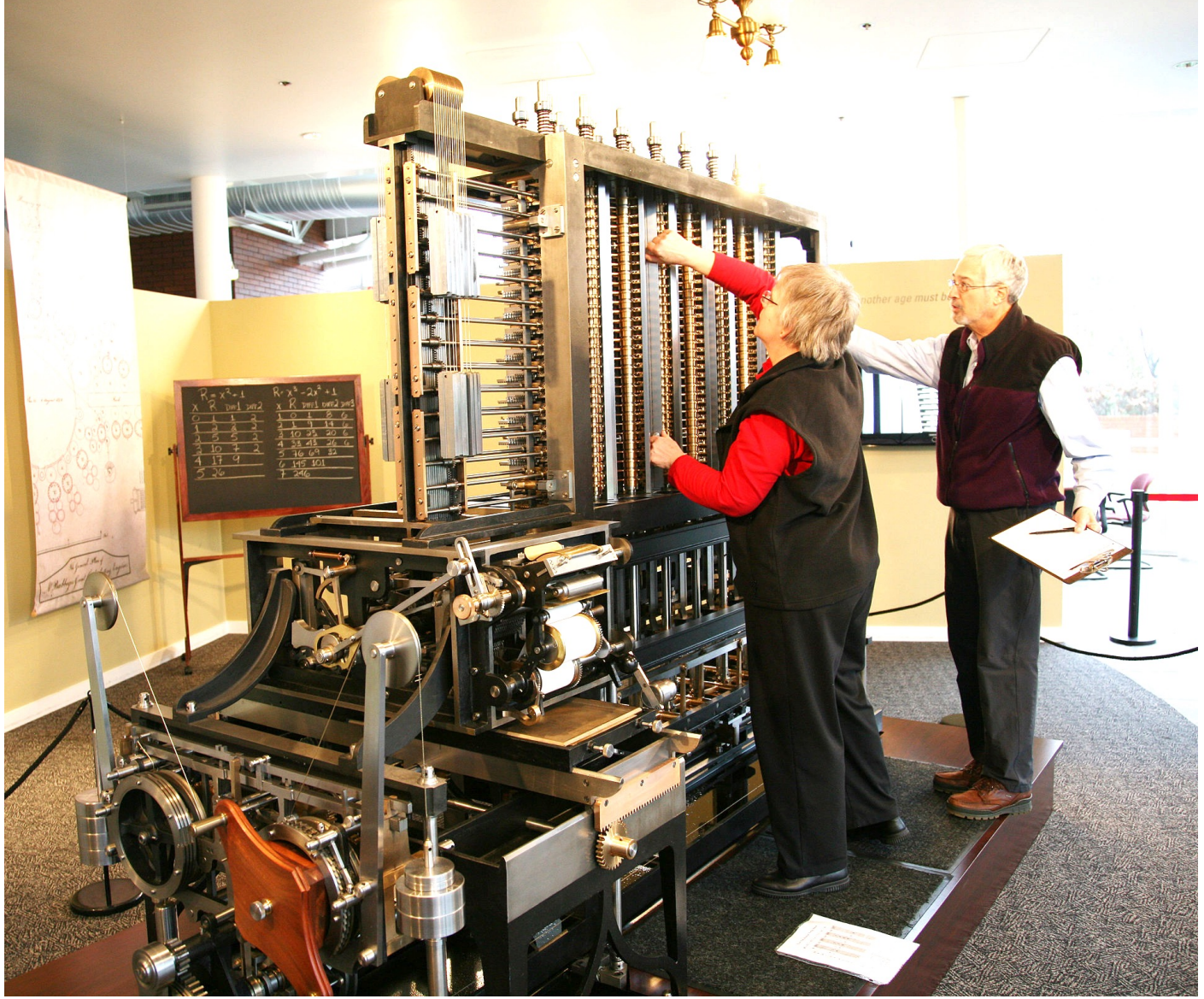
program *noun* [C] (ACTIVITIES)

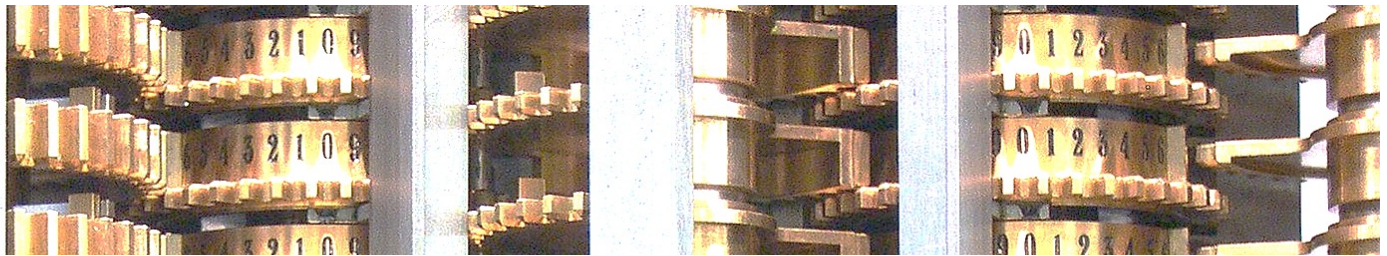
a group of activities or things to be achieved:

- *a training program*
- *the university basketball program*
- *a pilot recycling program*

Computer

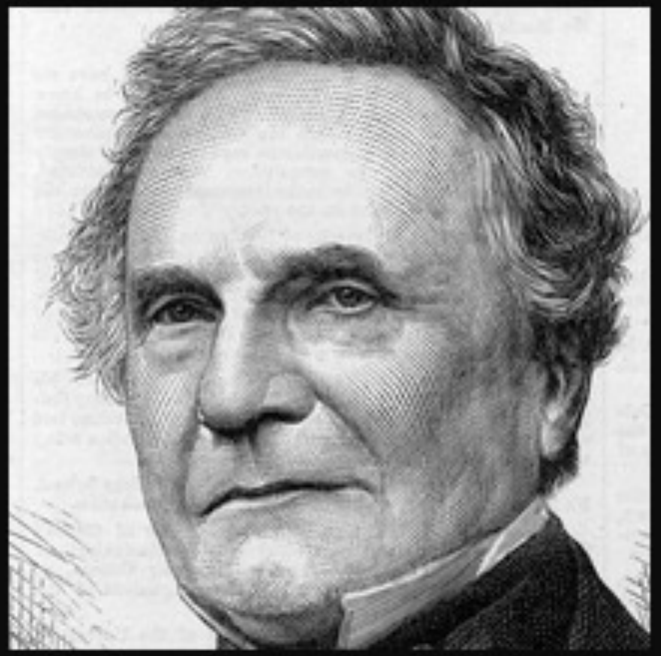
- **Mechanical** calculator
  - Processing numbers
  - Charles Babbage's "difference engine"
    - British mathematician, philosopher, inventor, mechanical engineer
    - Math professor at Cambridge University
  - Designed to tabulate/interpolate polynomial functions in 1820s
  - en.wikipedia.org/wiki/Difference_engine
  - en.wikipedia.org/wiki/File:Babbage_Engine_Demonstra





Computer

- On the importance of processing data (numerical, or in other forms)



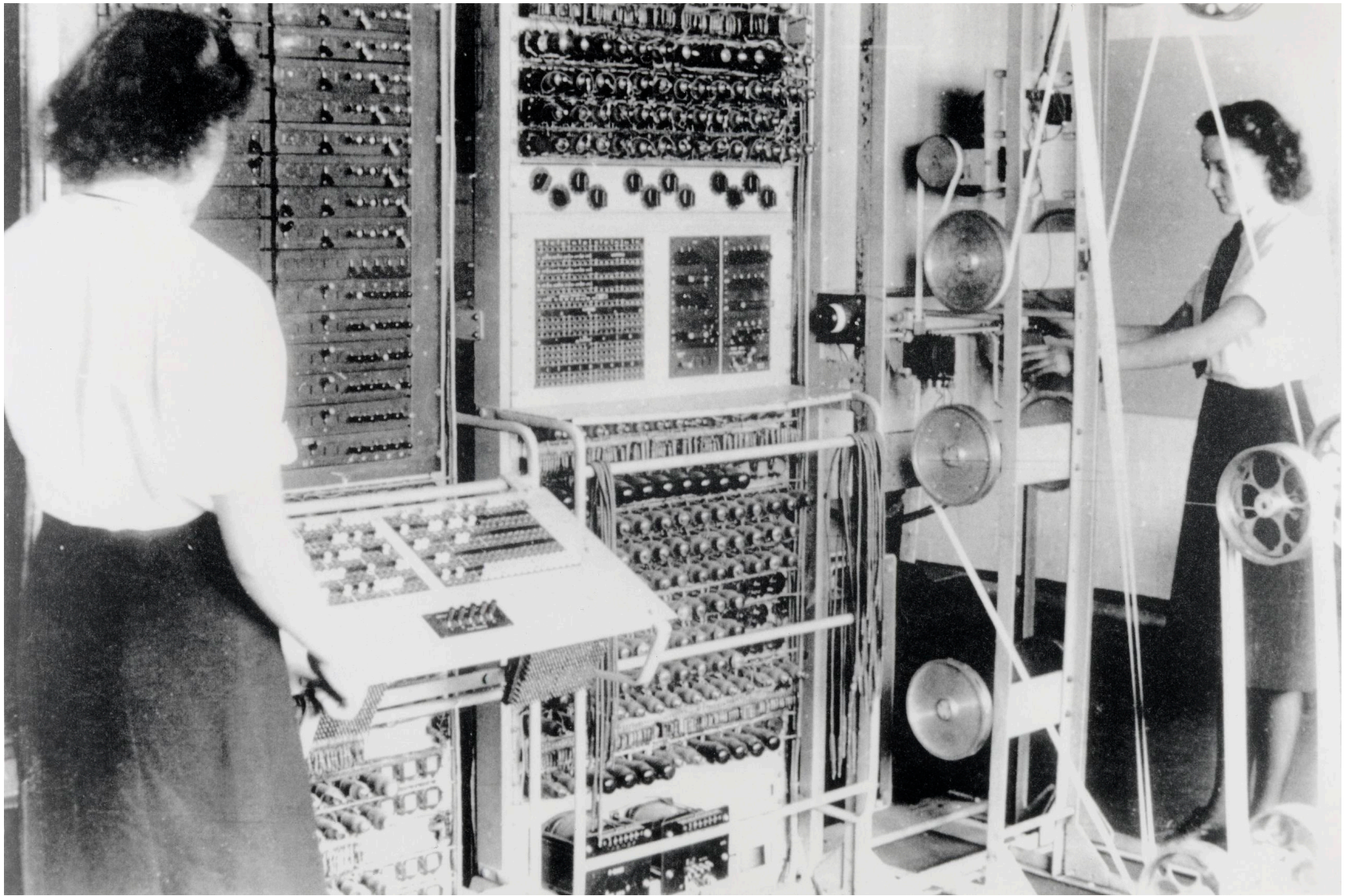
Errors using inadequate data are much less than those using no data at all.

~ Charles Babbage

AZ QUOTES

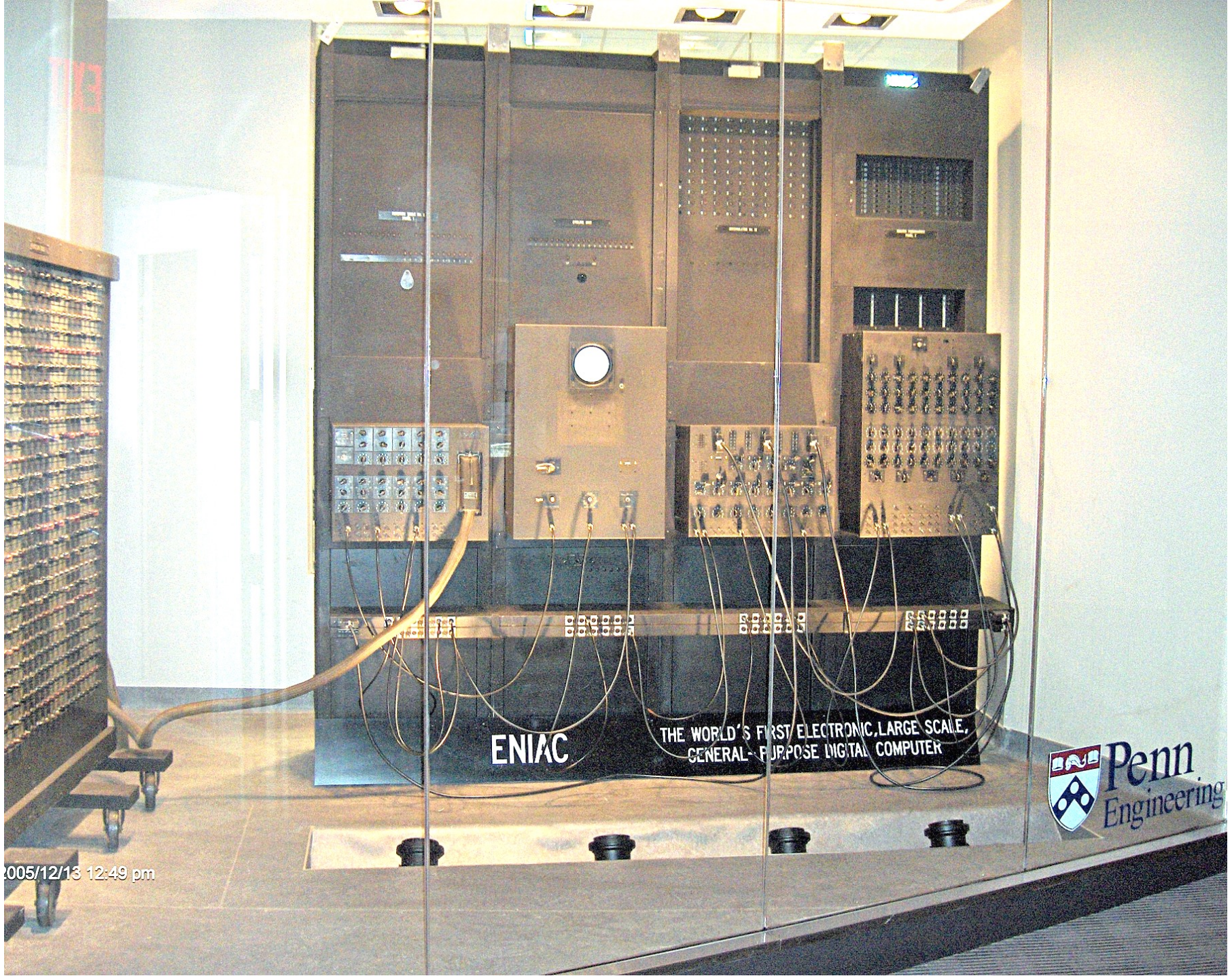
Computer

- **Electronic** digital computer
  - Cryptanalysis during second world war
  - Used thermionic valves (vacuum tubes) to perform logical and counting operations
  - en.wikipedia.org/wiki/Colossus_computer



Computer

- ENIAC (Electronic Numerical Integrator and Computer)
  - First programmable, electronic, **general-purpose** digital computer, completed in 1945
  - en.wikipedia.org/wiki/ENIAC



Computer

- Supercomputer
  - Modern
  - General-purpose computer with a high level of performance
    - en.wikipedia.org/wiki/Supercomputer

Computer

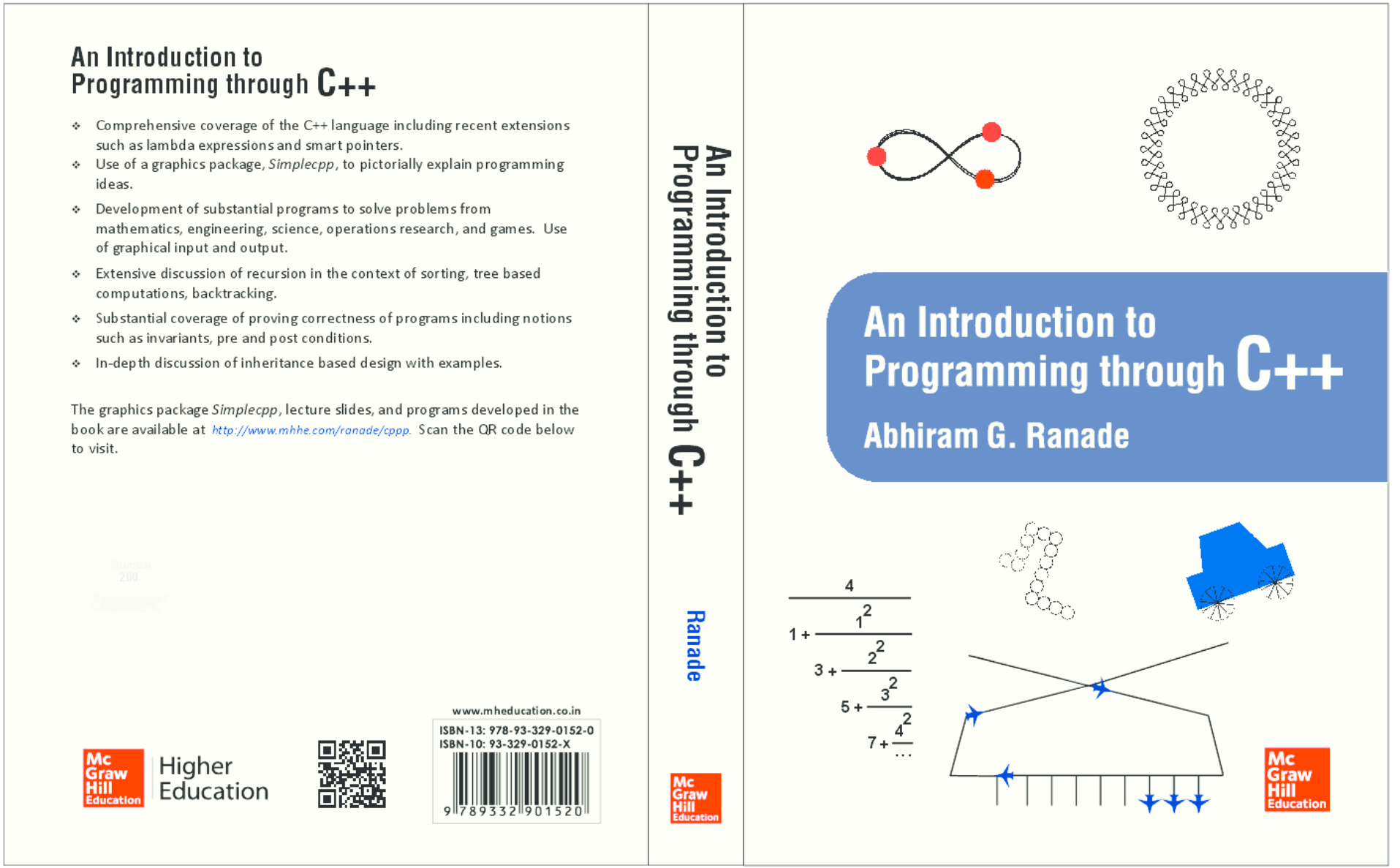
- Smartphone
  - “Portable computer device that combines mobile telephone functions and computing functions into one unit”
    - en.wikipedia.org/wiki/Smartphone

Programming

- "Computer program"
  - A sequence or set of instructions that the computer performs/executes/runs
  - A computer program in its human-readable form is called "source code", or simply "code"
- "Programming" (in the CS-101 context)
  - Process of performing particular computations (or more generally, accomplishing specific computing results), usually by designing and building executable computer programs
- "Programming language"
  - A system of notation for writing computer programs
  - Involves communication of a human with a computer
  - Aspects of "syntax" of language: form (how words are put together)
  - Aspects of "semantics" of language: meaning (of phrases and sentences)

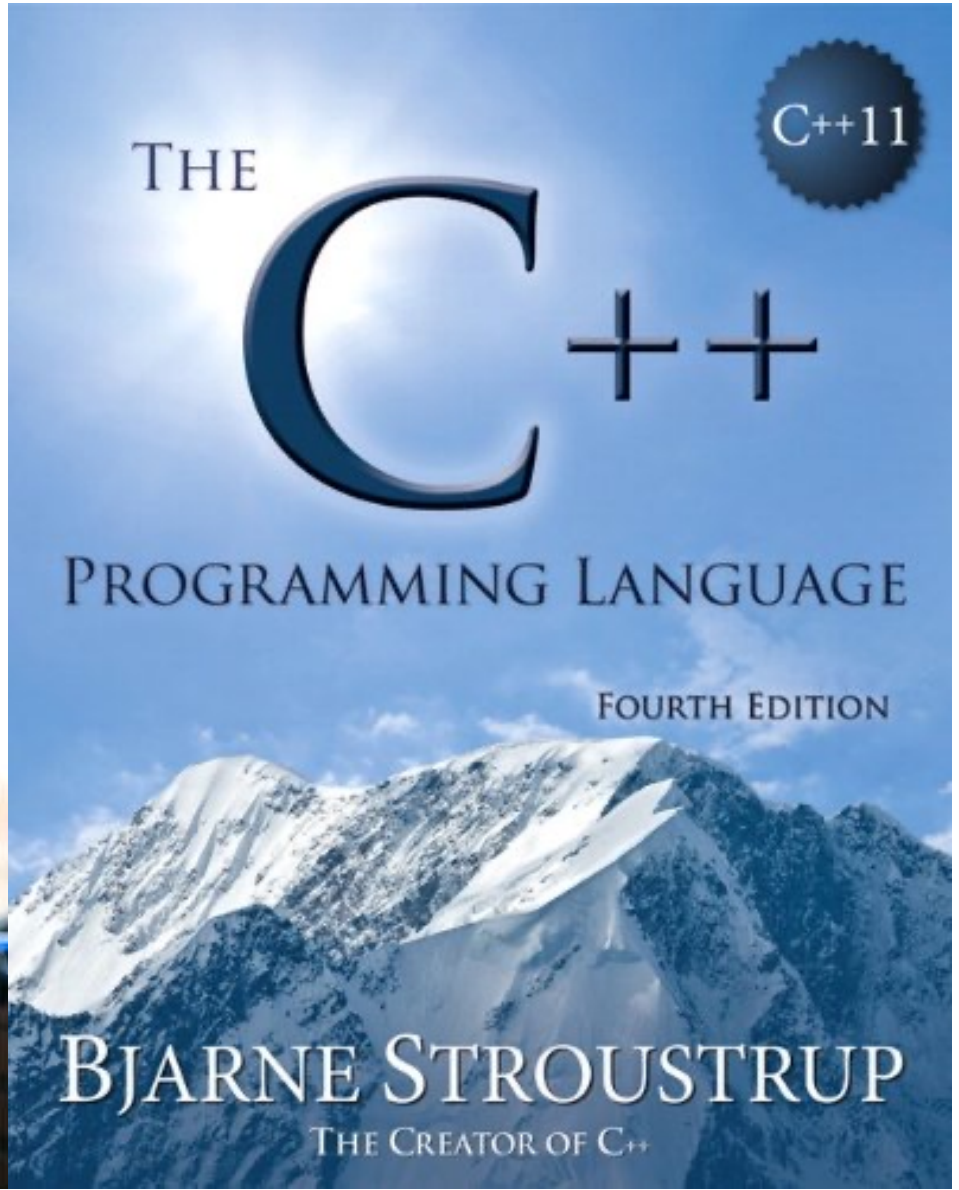
Main Text Book for CS-101

- www.cse.iitb.ac.in/~ranade/book.html
- Designed essentially for CS-101 students at IITB



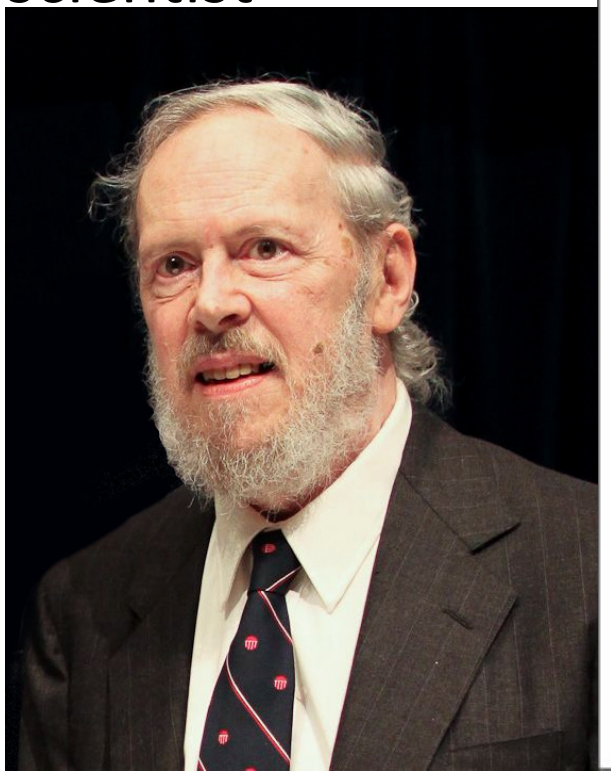
C++

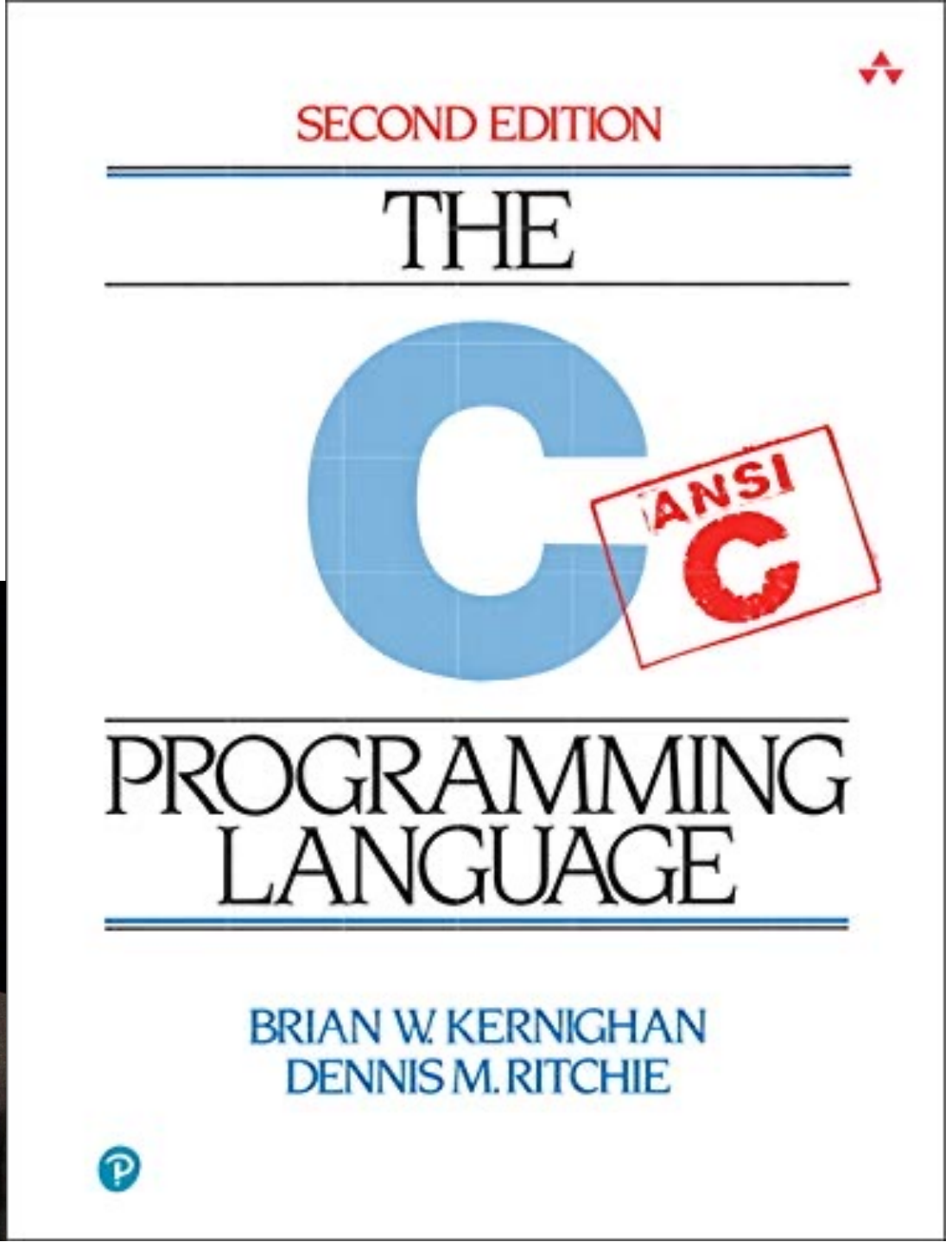
- High-level, general-purpose programming language
  - Created by Danish computer scientist Bjarne Stroustrup
    - PhD from University of Cambridge
    - Currently professor of computer science at Columbia University
  - Created in 1980s at AT&T Bell Labs
  - en.wikipedia.org/wiki/C++



C

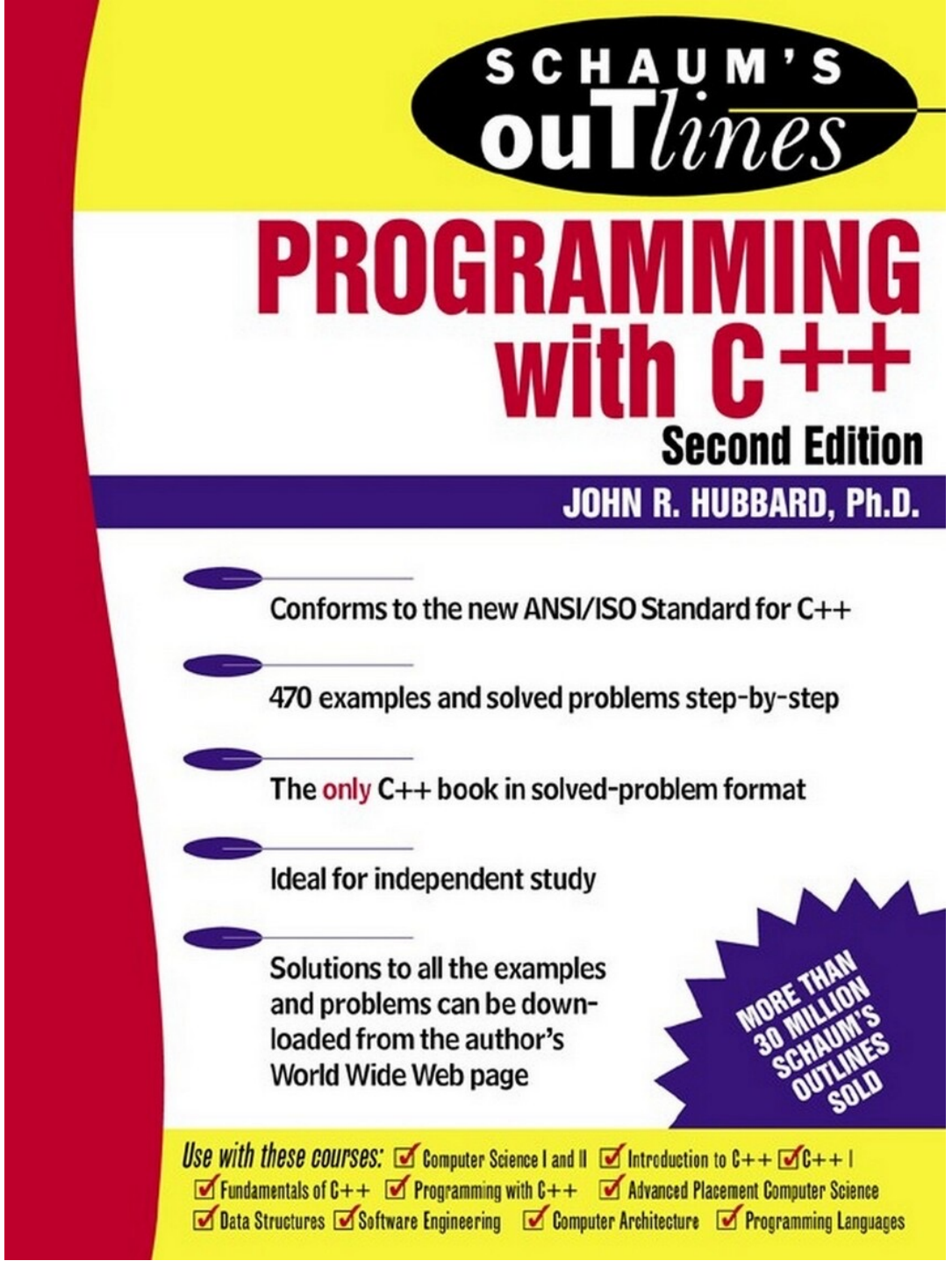
- Precursor was "C" programming language
  - Also a general-purpose programming language
  - Created in the 1970s by Dennis Ritchie, an American computer scientist (Turing Awardee)
    - PhD from Harvard Univ
  - C was successor to B
  - Among most widely used programming languages
  - Wikipedia links to books and more





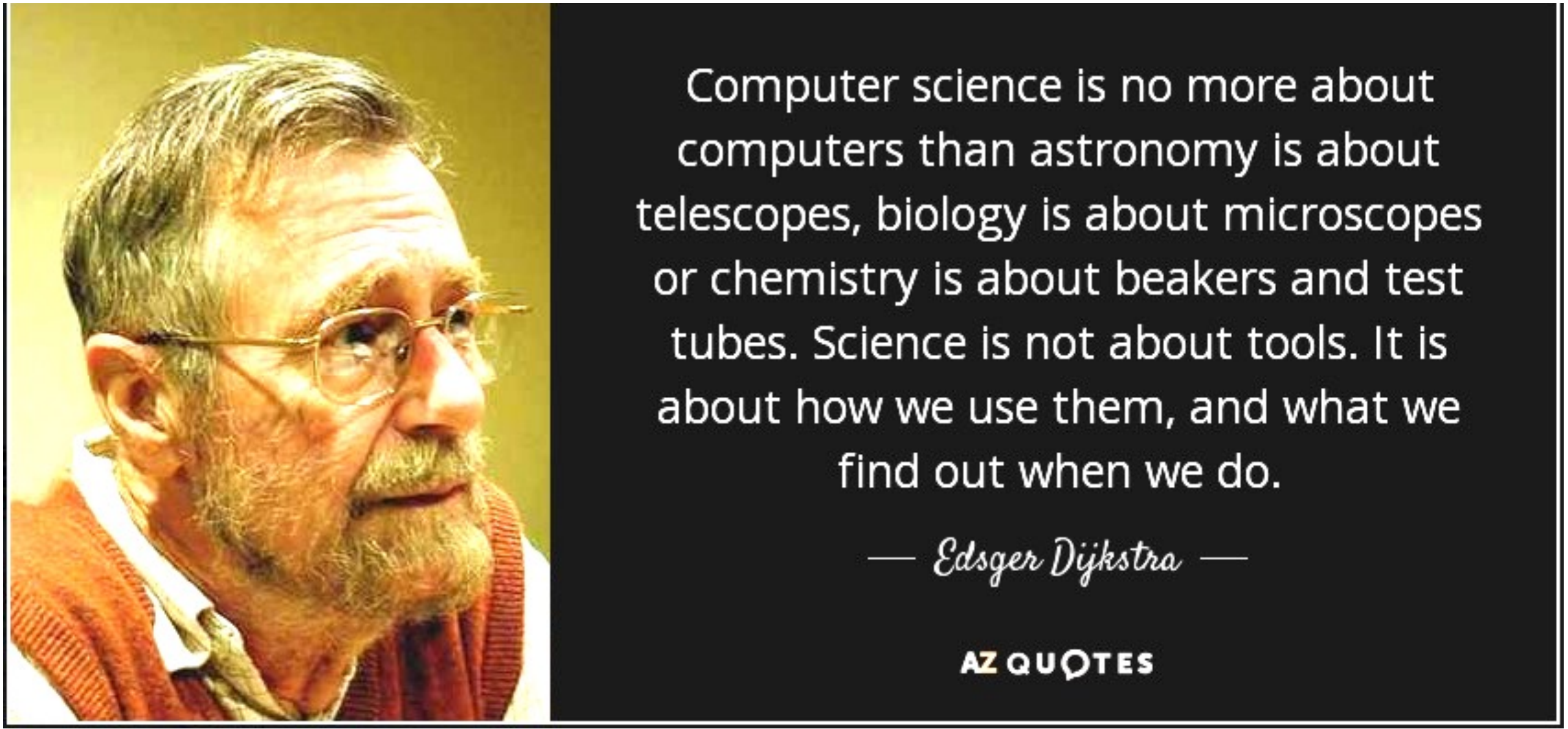
Reference Book for CS-101

- Remember: CS-101 isn't a course about any specific programming language, e.g., C++ or C
  - Our goal isn't to master all subtleties in any specific programming language
- This course is about fundamental concepts in programming, which are common to many languages



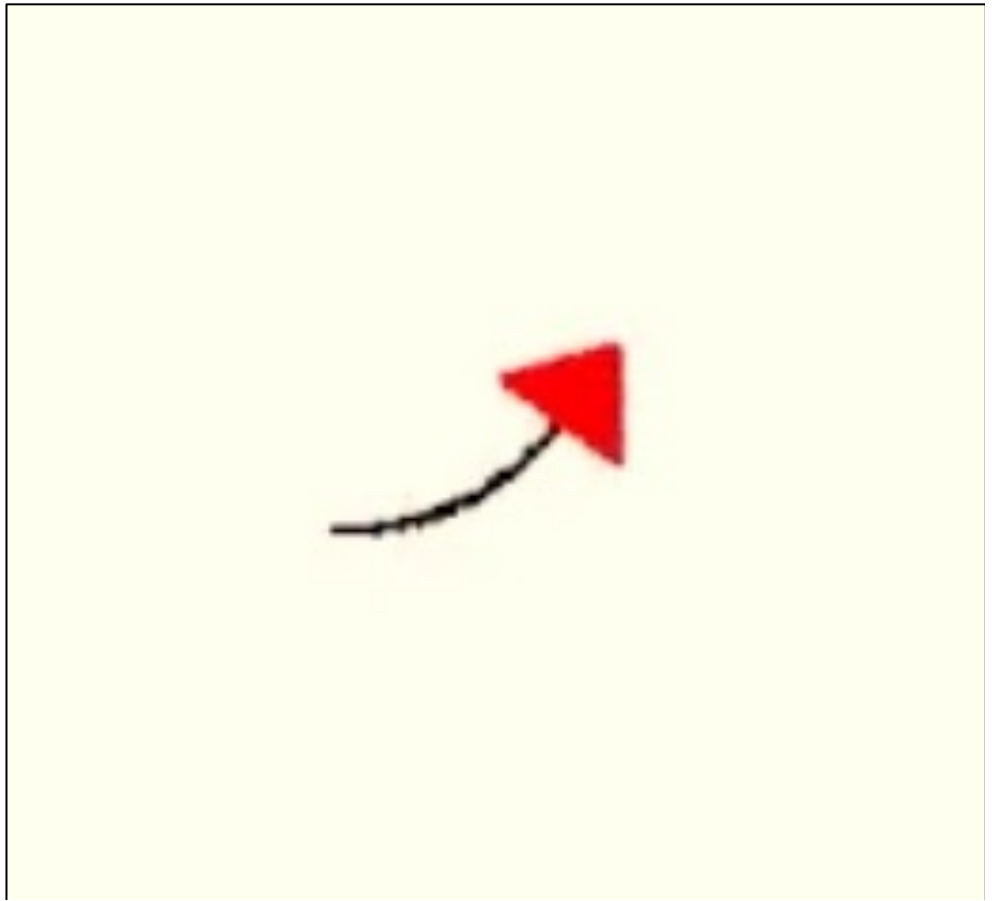
Computer Programming and CSE

- CS isn't only about computers or programming or languages. In fact, …



SimpleCPP

- SimpleCPP package is a gentler interface to C++
  - https://www.cse.iitb.ac.in/~ranade/simplecpp/
  - It has been designed specifically for the book
- Unlike C++, SimpleCPP has a built-in graphics interface
  - Called “turtleSim”,
    short form for “turtle simulator”
  - Allows you to move a “turtle”/cursor/pen-tip
  - “Turtle” can optionally draw as it moves
- We will start with SimpleCPP
  and then proceed to C++

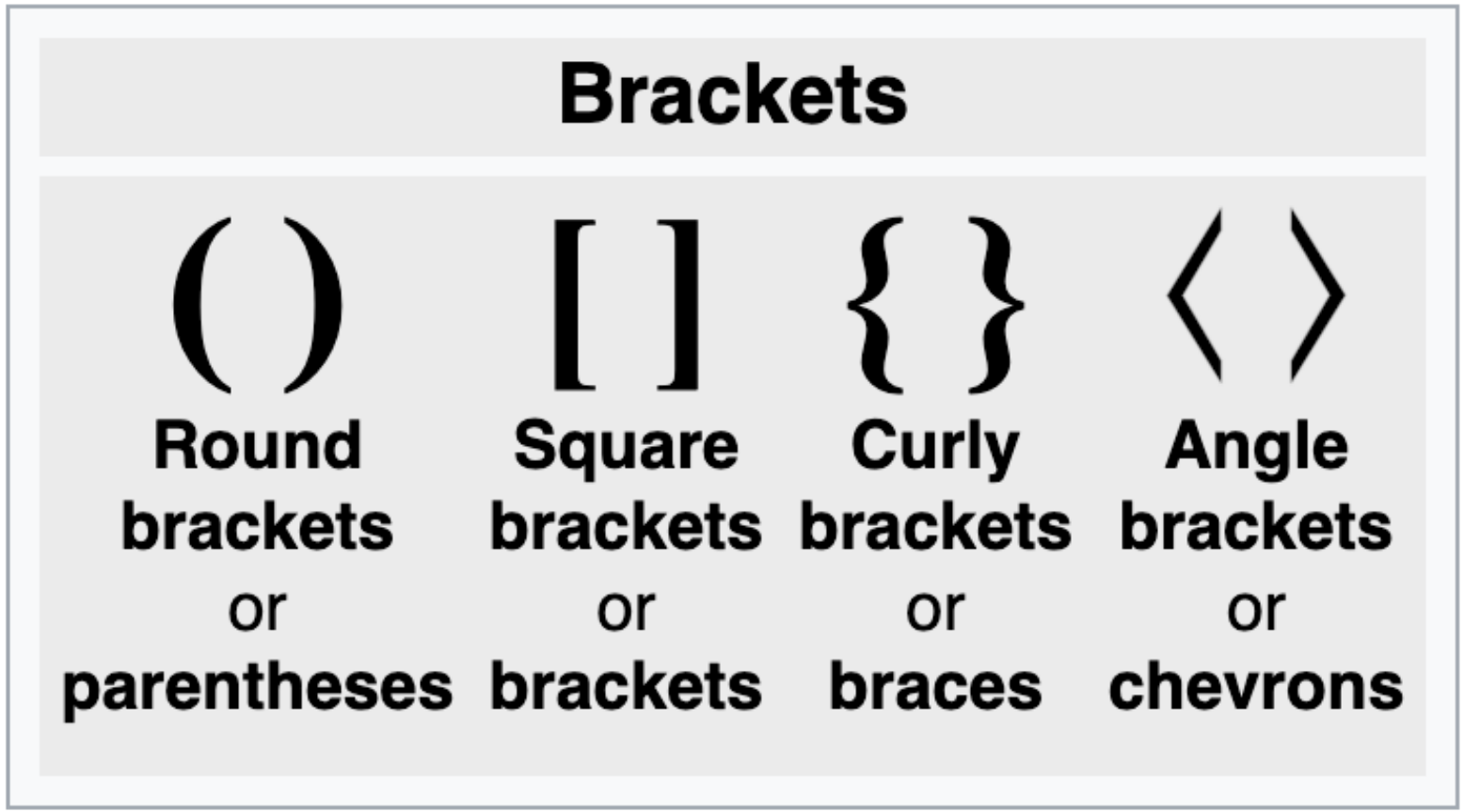


SimpleCPP

- History of the turtle
  - techcommunity.microsoft.com/t5/small-basic-bl
    small-basic-the-history-of-the-logo-turtle/ba-p/3
  - Turtle robots invented by William Grey Walter, a robotician, in 1948
  - No screens. So write.
  - en.wikipedia.org/wiki/Logo_(programming_language), 1967

First Program

- Lots of things going on; lets study one by one
- Some aspects of syntax
  - Brackets are of many kinds: ⟨ ⟩, { }, [ ], ( )
  - en.wikipedia.org/wiki/Bracket




```
#include <simplecpp>
main_program{
   turtleSim();

   forward(100);
   left(90);
   forward(100);
   left(90);
   forward(100);
   left(90);
   forward(100);

   wait(5);
}
```

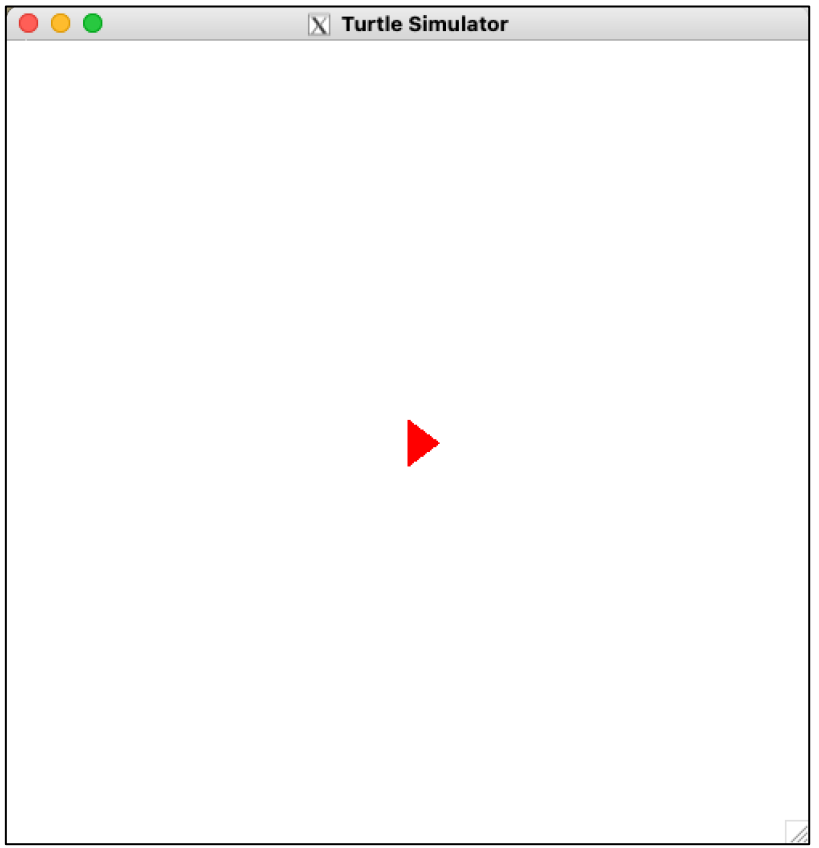
First Program

- #include …
  - Declares that program uses some facilities called “simplecpp” outside default facilities provided by C++
  - The program uses these facilities
    - Without this line, program won’t work
- main_program{ … }
  - A container of instructions/commands/statements
  - Inside the braces are the set of instructions the computer should start executing
  - There can be other containers too

```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

First Program

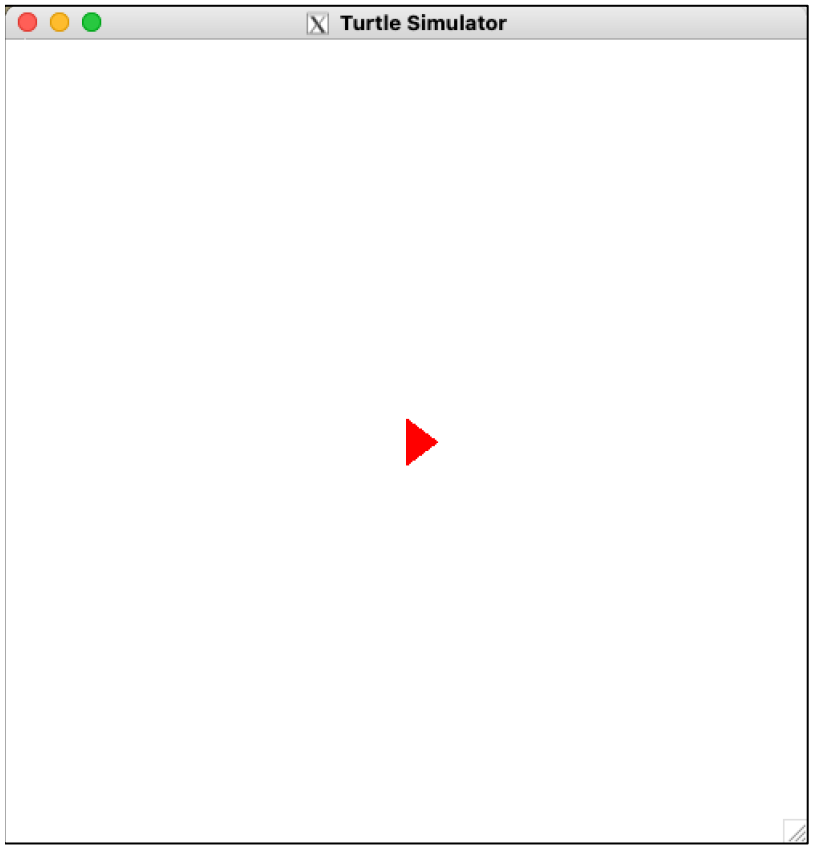
- turtleSim() command
  - Opens a window with a triangular cursor
  - Triangle = “turtle” with a pen that can be used to draw
  - Initially, turtle it directed towards east

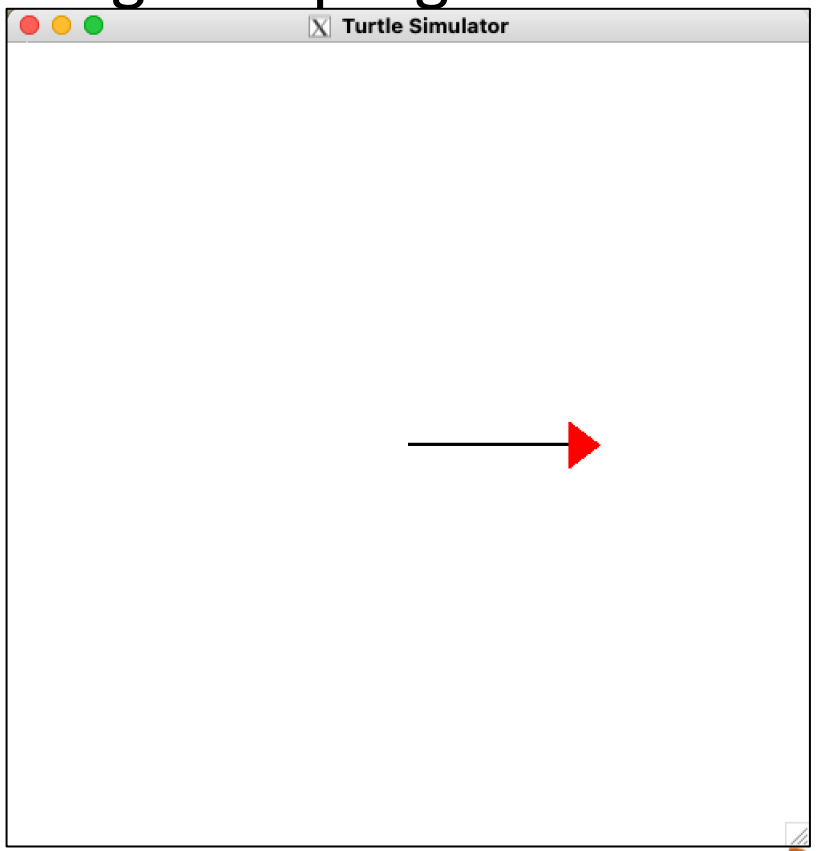



```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

First Program

- forward(100) command
  - Makes turtle move ahead (in current direction)
  - If pen state is “down”, turtle draws as it moves
  - Amount “100” can be changed in program

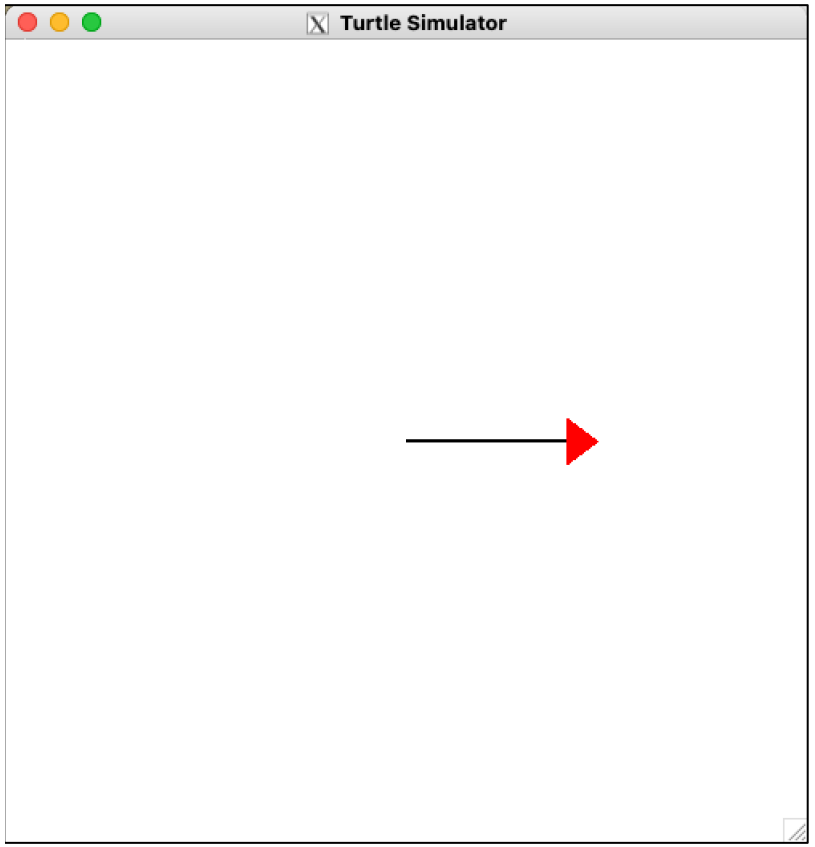


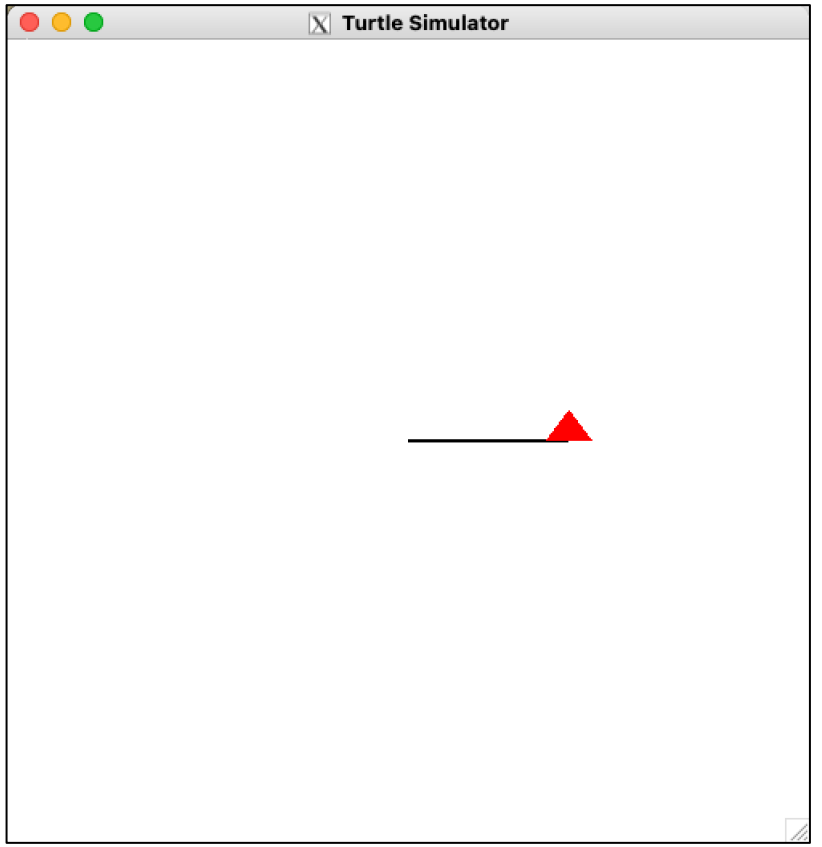



```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

First Program

- left(90) command
  - Changes direction by 90 degrees counter-clockwise
  - Amount “90” can be changed in program

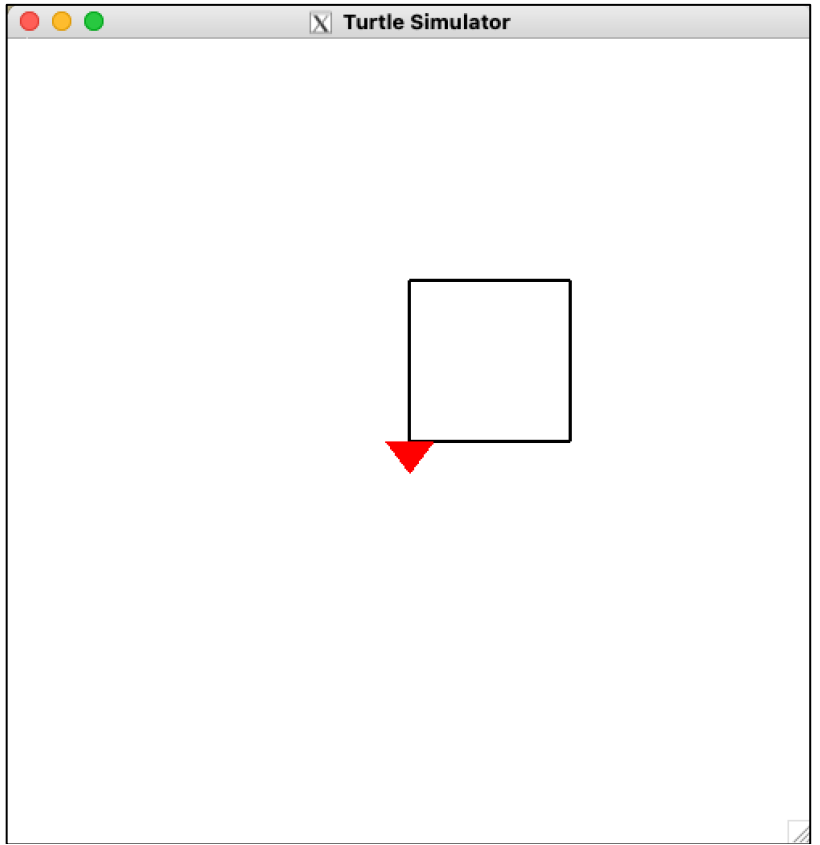





```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

First Program

- wait(5) command
  - Makes program execution wait for 5 seconds
- After the last line is executed, program ends
- "turtleSim" command doesn't need additional information, unlike forward() and left() commands. Hence, empty parentheses




```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

First Program

- Syntax
  - Each command must be followed by a semi colon
  - Programming languages usually allows spaces and linebreaks at many places
    - E.g., the following is valid, but not recommended

```
turtleSim();forward(100)    ;
left  (90
);
```

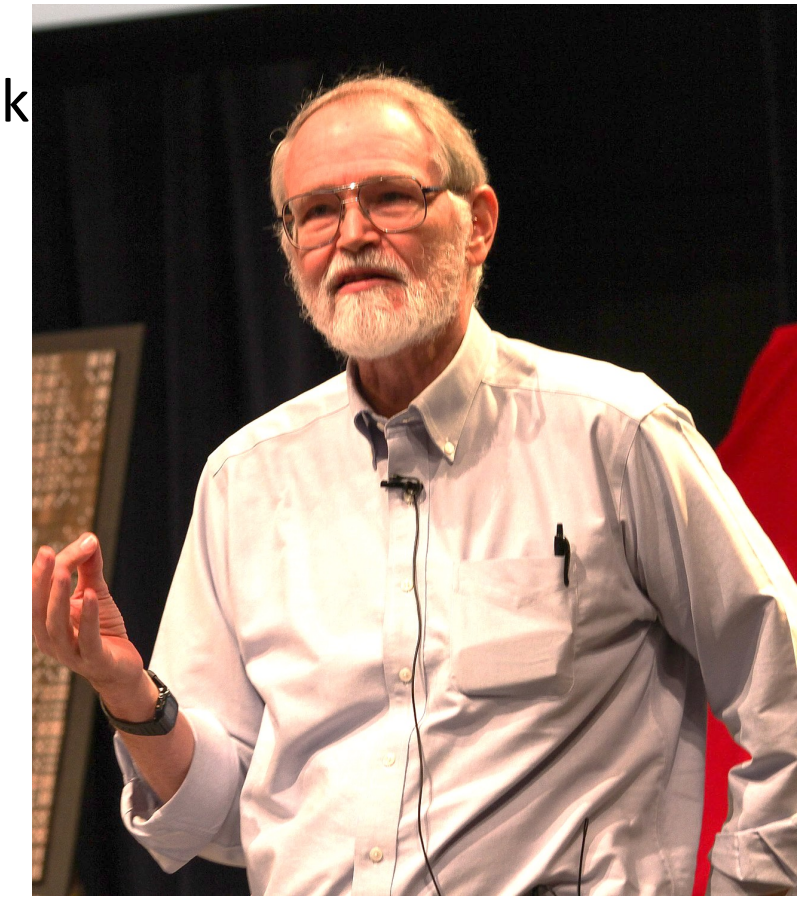
  - Indentation
    - e.g., putting leading spaces before lines for ease of readability
    - Designates “paragraphs” or logical blocks of commands within a program, to convey the structure within a program
    - Editors often help automate this process as we type

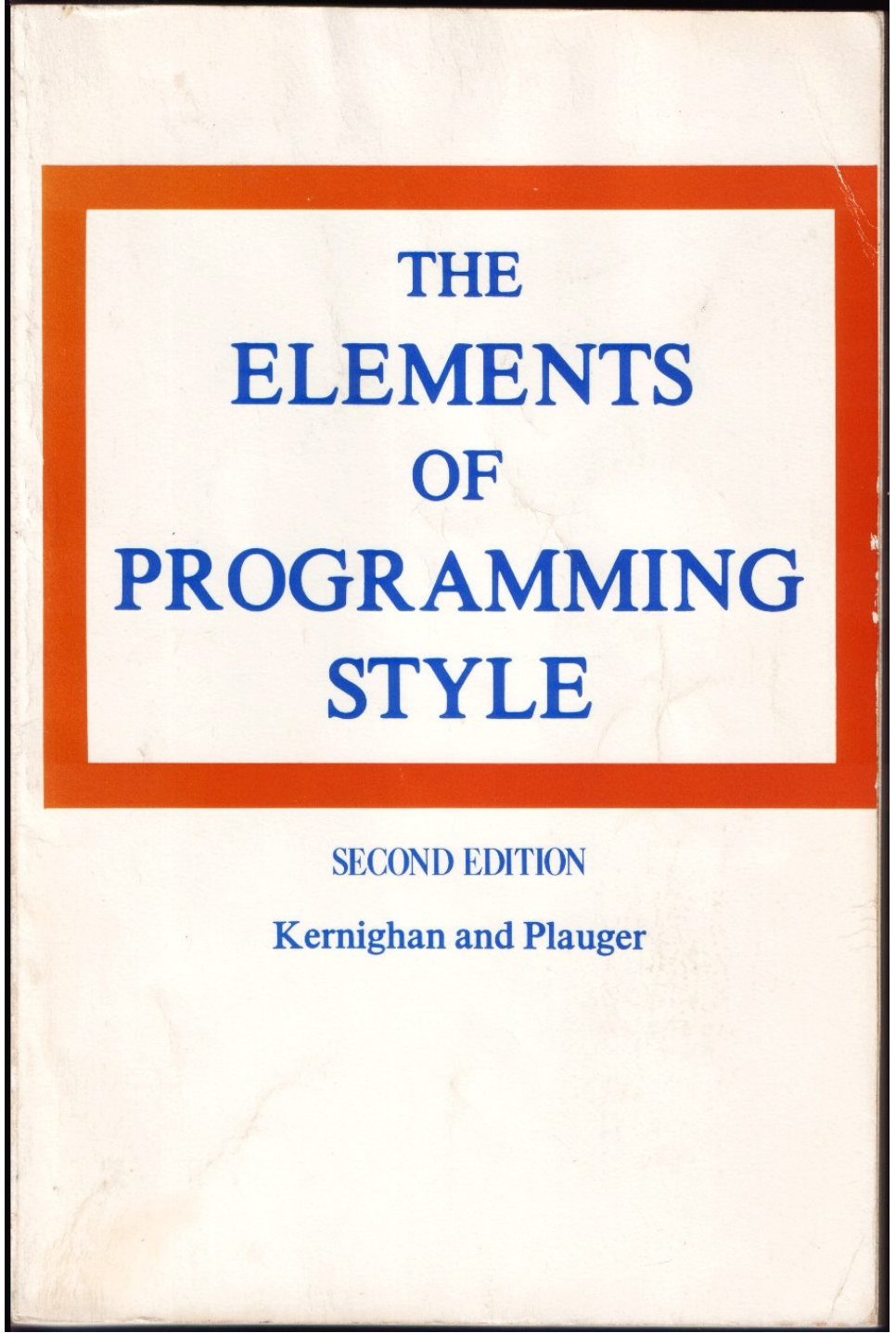
```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

Programming Style

Programming Style

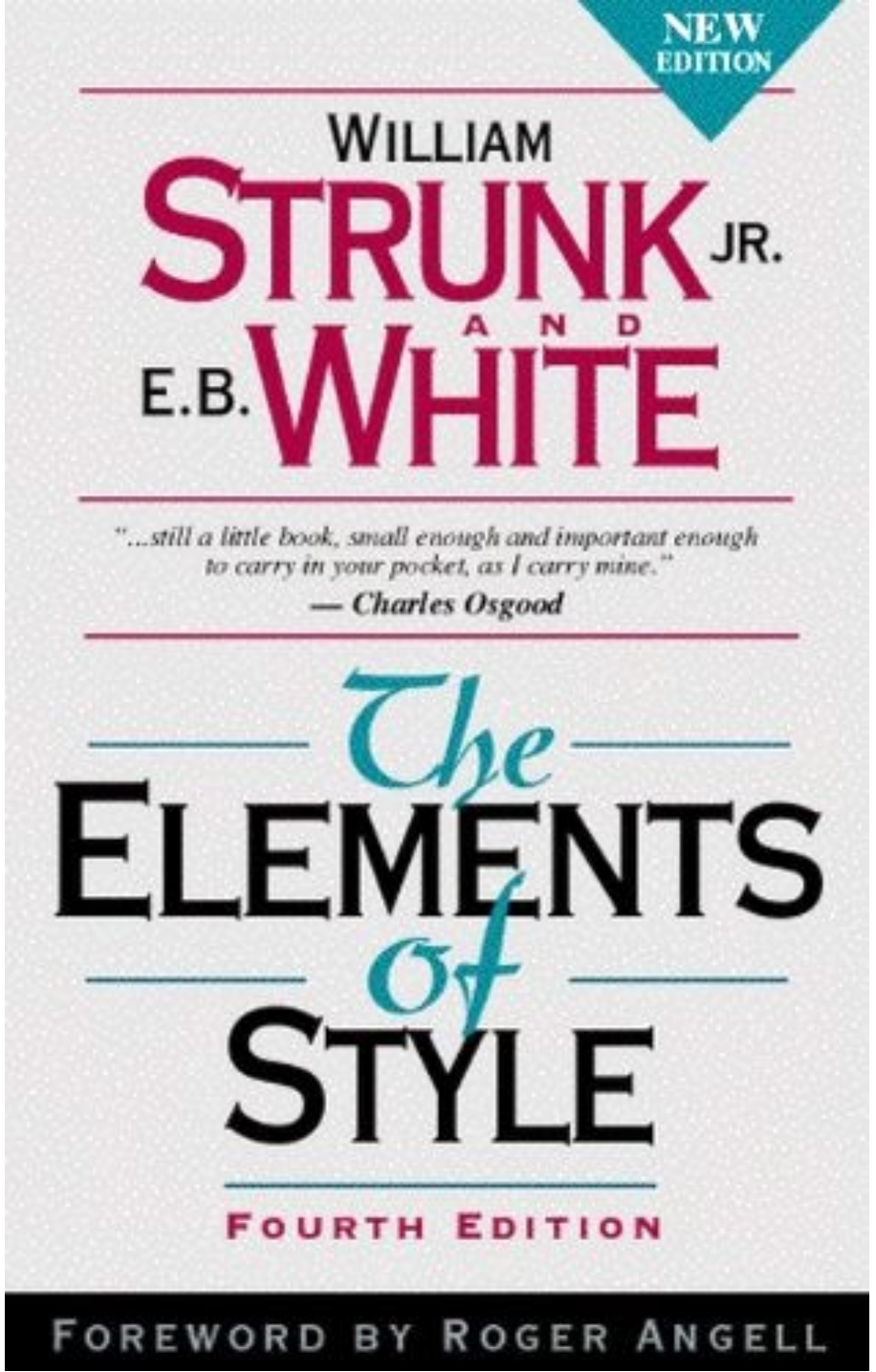
- The Elements of Programming Style
  - Brian W. Kernighan and P. J. Plauger
  - en.wikipedia.org/wiki/The_Elements_of_Programming_Style
    - Brian Kernighan
    - Co-wrote classic book on C programming with Dennis Ritchie (inventor of C) at Bell Labs



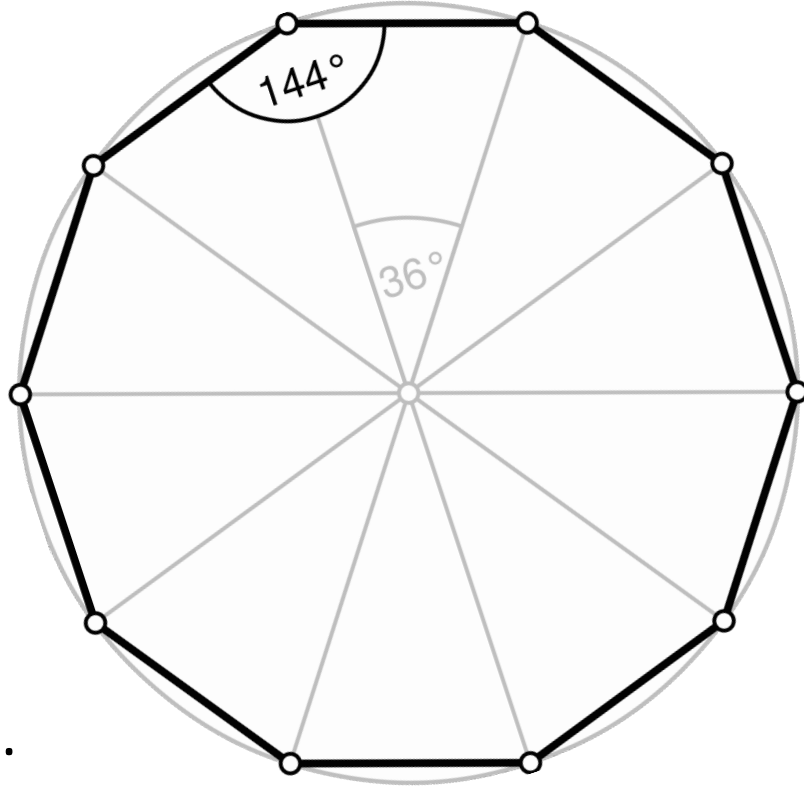


Technical-Writing Style

- The Elements of Style
  - William Strunk Jr. and E. B. White
  - First written by Strunk, professor of English at Cornell, in 1918
  - Then enlarged by his former student, White in 1959
  - Time magazine recognized this book, in 2011, as one of the 100 best and most influential books written in English since 1923



Programming Process

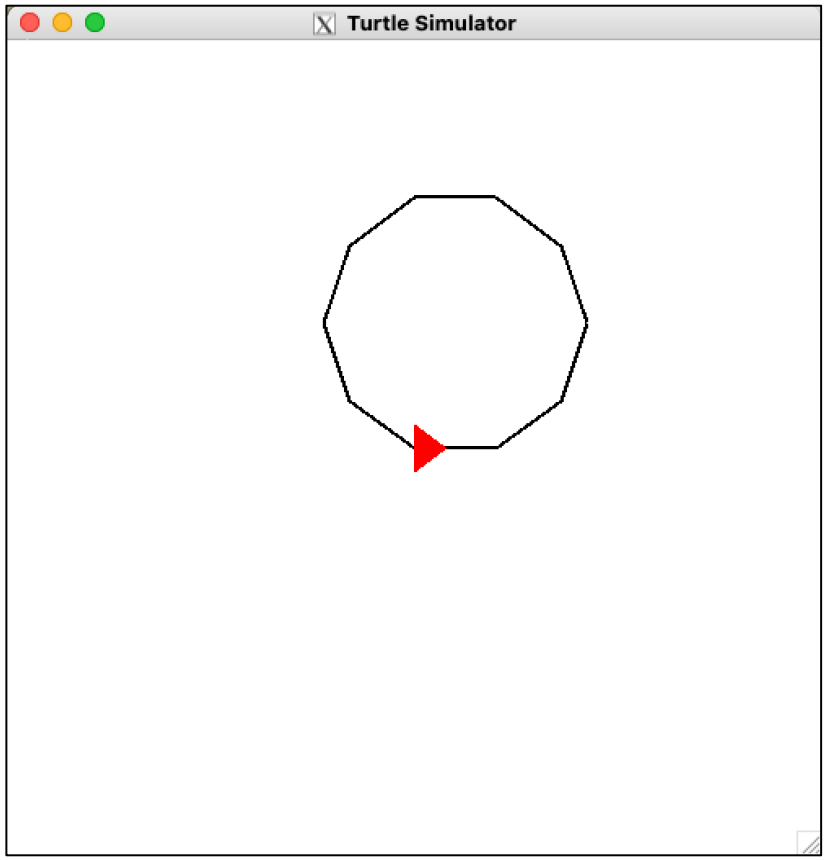


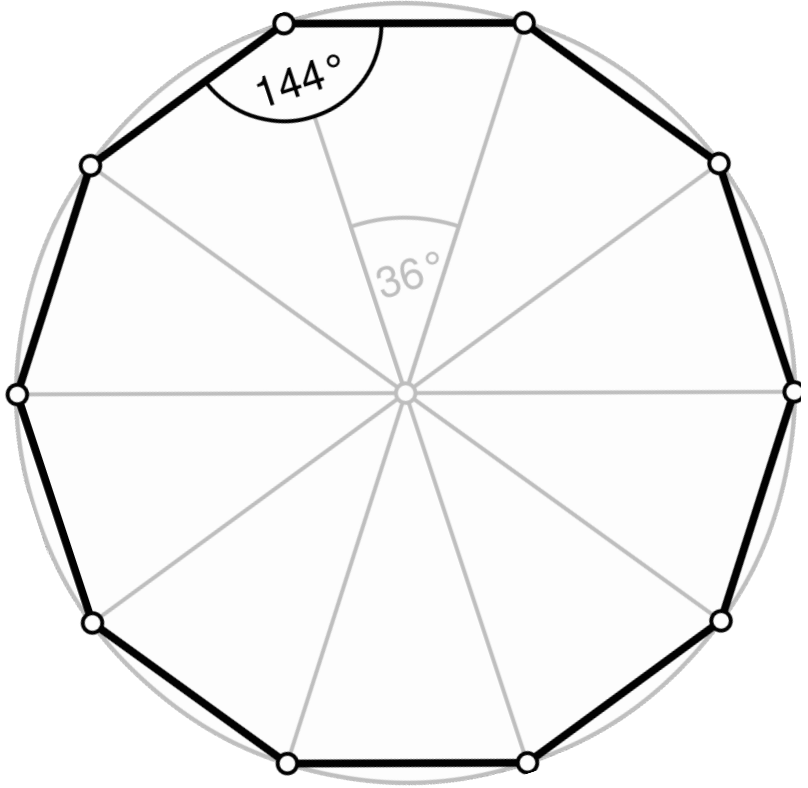
- Writing a program to draw a regular decagon
  - First thing to do is the math and logic **on paper**
    - Don't rush to start typing the program
  - After figuring out math and logic
    you may want to write a (rough) program **on paper**
  - Then run program **in your head**
    - Find problems with (rough) program and fix them. Repeat.
  - Then start typing to input program into computer
  - As you've typed a part of program, you may want to run that part
    to ensure that it works as expected
    - Otherwise there is some error in your math or logic (or typing/syntax). If so, fix it.
  - Finish writing your program and run it
    - Check output
    - If there is some error in your math or logic, revisit typed program to fix error

Repeating a Block of Commands

- Writing a program to draw a regular decagon

```
#include <simplecpp>
main_program{
   turtleSim();
   repeat(10){
         forward(100);
         left(36);
   }
   wait(5);
}
```



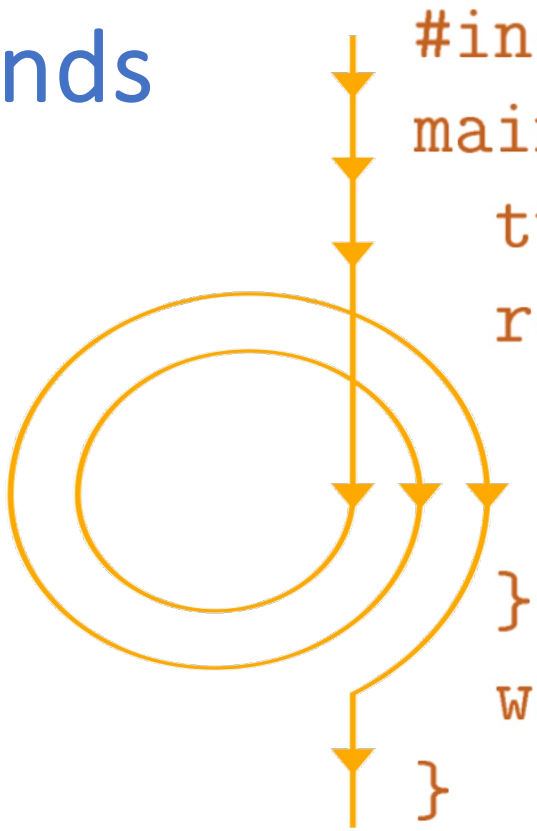


- Repeat statement's general form
  - Statements = "body" of repeat statement/command
  - Each execution of body is called an "iteration"

```
repeat(count){
   statements
}
```

Repeating a Block of Commands

- What we have is a “repeat loop”
- “Loop” = control-flow statement for specifying “iteration”



```
#include <simplecpp>
main_program{
   turtleSim();
   repeat(10){
        forward(100);
        left(36);
   }
   wait(5);
}
```

iteration

noun [C or U]

UK /ˌɪt.ᵊrˈeɪ.ʃᵊn/ US /ˌɪt̬.əˈreɪ.ʃᵊn/

Add to word list

formal

the process of doing something again and again, usually to improve it, or one of the times you do it:

- *the repetition and iteration that goes on in designing something*

Repeating a Block of Commands

- Writing a program to allow user to draw **any** regular polygon

1. Need to ask user
   how many sides does the polygon have
2. Need to get input from the user
   into the program
3. Need to use
   that information
   into our
   drawing procedure

```
#include <simplecpp>
main_program{
  turtleSim();
  repeat(10){
        forward(100);
        left(36);
  }
  wait(5);
}
```

```
#include <simplecpp>
main_program{
  turtleSim();

  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);
  left(90);
  forward(100);

  wait(5);
}
```

Repeating a Block of Commands

- Writing a program to allow user to draw any regular polygon
  - "int" indicates integer
  - "int nsides" makes computer reserve a region in its memory named 'nsides' which will store an integer value
    - Future references by program to 'nsides' will refer to content stored (= value) in this memory region
  - 'nsides' is called a "variable"
  - "int nsides" defines/creates variable 'nsides'

```
#include <simplecpp>
main_program{
   int nsides;
   cout << "Type in the number of sides: ";
   cin >> nsides;
   turtleSim();
   repeat(nsides){
         forward(50);
         left(360.0/nsides);
   }
   wait(5);
}
```

Repeating a Block of Commands

- Writing a program to allow user to draw any regular polygon
  - 'cout' is a name that refers to computer display/screen
  - 'cout <<' indicates something being passed on to cout, which is denoted within "…"
  - 'cin' refers to keyboard
  - 'cin >>' indicates getting something from keyboard and storing it in memory region indicated by variable following it

```
#include <simplecpp>
main_program{
   int nsides;
   cout << "Type in the number of sides: ";
   cin >> nsides;
   turtleSim();
   repeat(nsides){
         forward(50);
         left(360.0/nsides);
   }
   wait(5);
}
```

Repeat Within a Repeat

- What does this do ?
  - What if nsides = 2 ?
  - What if nsides = 1 ?
  - What if nsides = 0 ?
  - What if nsides < 0 ?
  - Duty of the program to verify correctness of input

```
#include <simplecpp>
main_program{
  int nsides;

  turtleSim();

  repeat(10){
    cout << "Type in the number of sides: ";
    cin >> nsides;
    repeat(nsides){
        forward(50);
        left(360.0/nsides);
    }
  }
  wait(5);
}
```

Garbage In Garbage Out

- en.wikipedia.org/wiki/Garbage_in,_garbage_out

On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" ... I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

—Charles Babbage, *Passages from the Life of a Philosopher*[5]

More Commands in SimpleCpp

- penUp(), penDown()
- What will the following code do ?
  - repeat (10)
    { forward (10); penUp(); forward (5); penDown(); }
- Numerical functions
  - sqrt(x)
  - sine(x), cosine(x), tangent(x) – where ‘x’ is in degrees
  - sin(x), cos(x), tan(x) – where ‘x’ is in radians
  - arcsine(y), arccosine(y), arctangent(y)
  - exp(x), log(x), log10(x)
  - pow(x,y)
  - Name ‘PI’ refers to π

Practice Examples for Lab: Set 1

- 1 done

Modify the program given in the text so that it asks for the side length of the polygon to be drawn in addition to asking for the number of sides.

- 2 done

A pentagram is a five pointed star, drawn without lifting the pen. Specifically, let A,B,C,D,E be 5 equidistant points on a circle, then this is the figure A–C–E–B–D–A. Draw this.

- 3 done

If you draw a polygon with a large number of sides, say 100, then it will look essentially like a circle. In fact this is how circles are drawn: as a many sided polygon. Use this idea to draw the numeral 8 – two circles placed tangentially one above the other.

- 4 done

Read in the lengths of the sides of a triangle and draw the triangle. You will need to know and use trigonometry for solving this.

Practice Examples for Lab: Set 1

- 5 done

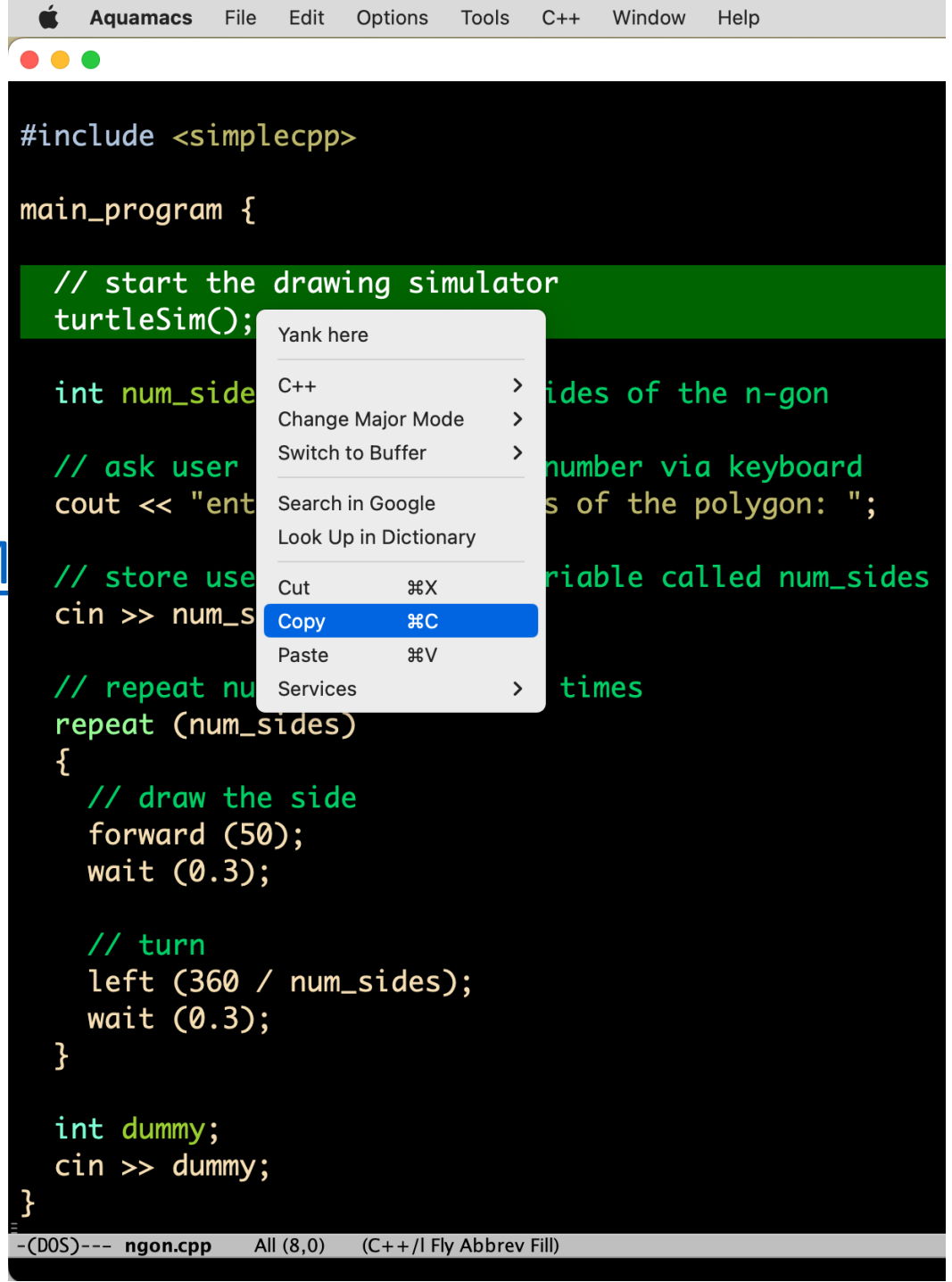
We wrote "360.0" in our program rather than just "360". There is a reason for this which we will discuss later. But you could have some fun figuring it out. Rewrite the program using just "360" and see what happens. A more direct way is to put in statements `cout << 360/11; cout << 360.0/11;` and see what is printed on the screen. This is an important idea: if you are curious about "what would happen if I wrote ... instead of ...?" – you should simply try it out!

- 6 done
  - https://en.wikipedia.org/wiki/Heptagram

Draw a seven pointed star in the same spirit as above. Note however that there are more than one possible stars. An easy way to figure out the turning angle: how many times does the turtle turn around itself as it draws?

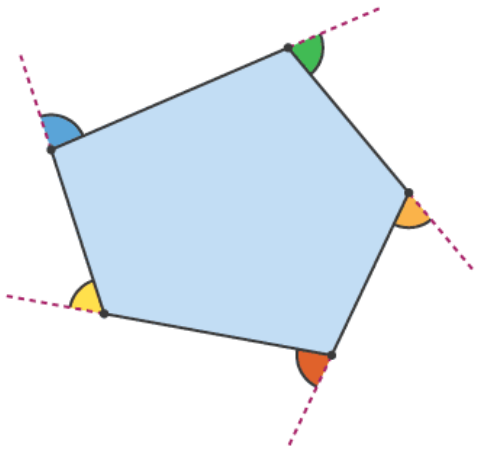
Emacs

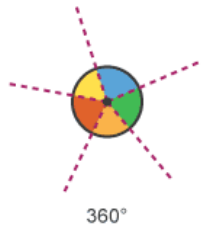
- Emacs on Ubuntu
  - 3 ways to install Emacs on Ubuntu linux.how2shout.com/3-ways-to-install-emacs-text-editor-on-ubuntu-22-04-lts/
  - How to Install EMACS on Ubuntu www.youtube.com/watch?v=aNpexBTEQBM
- Emacs on Windows
  - www.gnu.org/software/emacs/download.html
  - gnu.mirror.net.in/gnu/emacs/windows
    get the most-recent dated version
- Emacs on MacOS
  - aquamacs.org

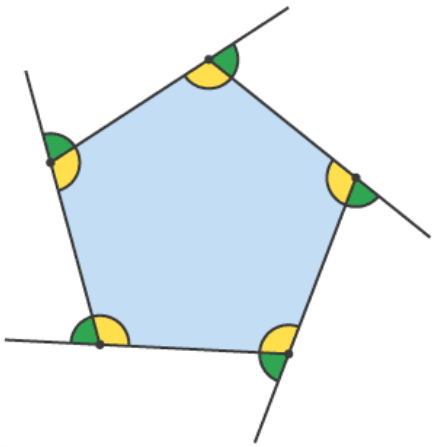


Comments

```
#include <simplecpp>
/* Program to draw a regular polygon with as many sides as the user wants.
   Author: Abhiram Ranade
   Date:   18 Feb 2013.
*/
main_program{
  int nsides;
  cout << "Type in the number of sides: ";
  cin >> nsides;

  turtleSim();

  repeat(nsides){
     forward(50);          // Each side will have length 50 pixels.
     left(360.0/nsides); // Because sum(exterior angles of a polygon) = 360.
  }
  wait(5);
}
```

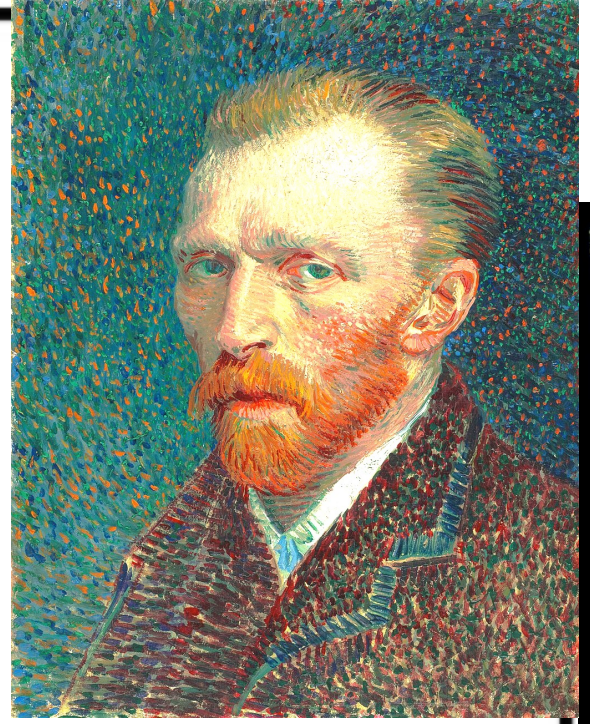






Comments

- Don’t state the obvious
  - Examples to avoid
    - cin >> nsides; // read in nsides
    - Above comment only states what is being done, but that is clear from syntax itself
- Give **insights** into the logic used
  - Describe **why** are you doing what you are doing
- Aim to write **self-documenting code**
  - Make the code read as close to English as possible
  - Chose names of variables wisely to make is obvious what their use is
  - Write down the logic
  - Minimize amount of "external" documentation that is necessary



PROGRAMMING IS AN ART

Comments

- www.youtube.com/watch?v=OlacUh

Comments

Funny Source Code Comments

```
//
// Dear maintainer:
//
// When I wrote this code, only I and God
// knew what it was.
// Now, only God knows!
//
// So if you are done trying to 'optimize'
// this routine (and failed),
// please increment the following counter
// as a warning
// to the next guy:
//
// total_hours_wasted_here = 67
//
```

Comments


fb.com/programmingjokes/
NERD LIFE studio
what makes code bad?
No comment.

Commen

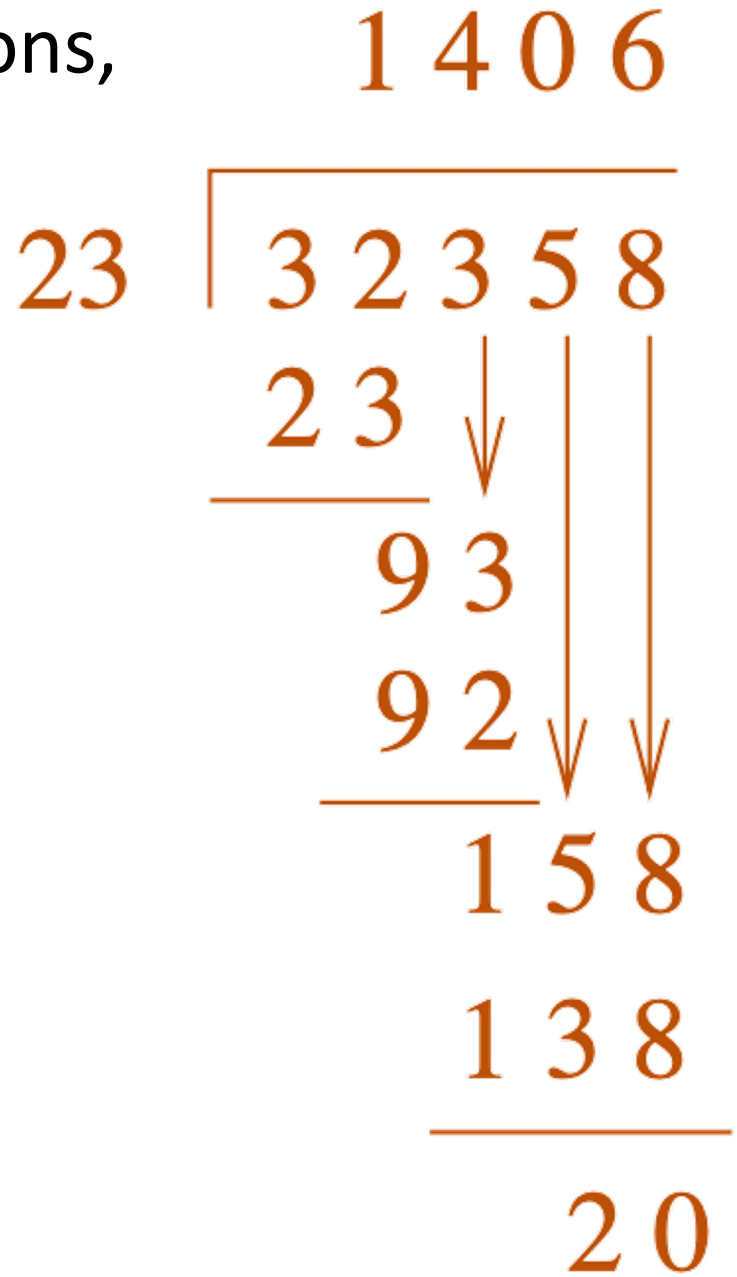

YEAR 0
YEAR X
LET ME WRITE THIS CODE IN ONE LINE WITH COMPLEX ABSTRACTIONS. I AM SO CLEVER!
LET ME WRITE THIS CODE SUCH THAT FUTURE-ME CAN EASILY UNDERSTAND IT.
workchronicles.com

Computation Without Graphics

```
main_program{
   int n;
   cout <<"Type the number you want cubed: ";
   cin >> n;
   cout << n*n*n << endl;
}
```

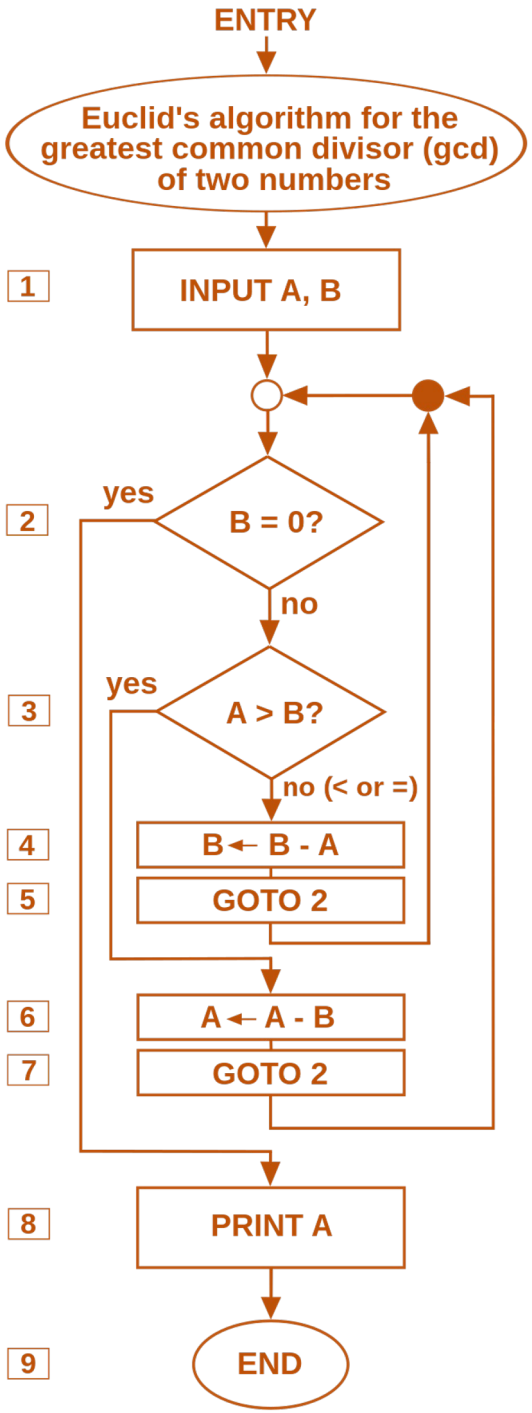
Algorithm

- Algorithm = a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation
  - en.wikipedia.org/wiki/Algorithm
- Example: Algorithm for long division
- Origin of the word
  - Around 825,
    Muhammad ibn Musa al-Khwarizmi
    wrote kitāb al-ḥisāb al-hindī
    (“Book of Indian computation”) and
    kitab al-jam' wa'l-tafriq al-ḥisāb al-hindī
    (“Addition and subtraction in Indian arithmetic”)

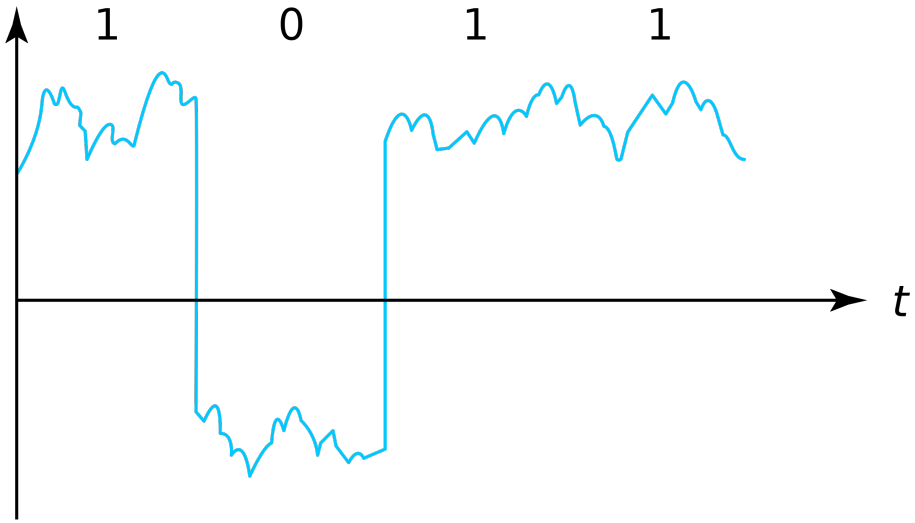


Algorithm

- Example:
  Euclid’s algorithm for finding greatest common divisor (GCD)



Digital Electronics



- "bit" denotes a number that is either 0 or 1
  - Like a "digit"
    that represents a number within 0,1,2,…,9
  - Number in binary number system
    (binary number) is represented using bits
  - Mapping voltages, in electric circuits, to highs and lows, or ones and zeros
    - Use this to represent a binary (base-2) number system
- Binary system in ancient India
  - Weights excavated from Indus Valley Civilization corresponded to ratios 1, 2, 4, 8, 16, …; basis of binary numbers, because ratios double
    - en.wikipedia.org/wiki/History_of_measurement_systems_in_India#Early_history
  - Pinagala created it in 3rd century BC to study poetic meters
    - https://en.wikipedia.org/wiki/Pingala#Combinatorics
    - www.jstor.org/stable/23445644 [Binary Numbers in Indian Antiquity]

Number Representation Formats

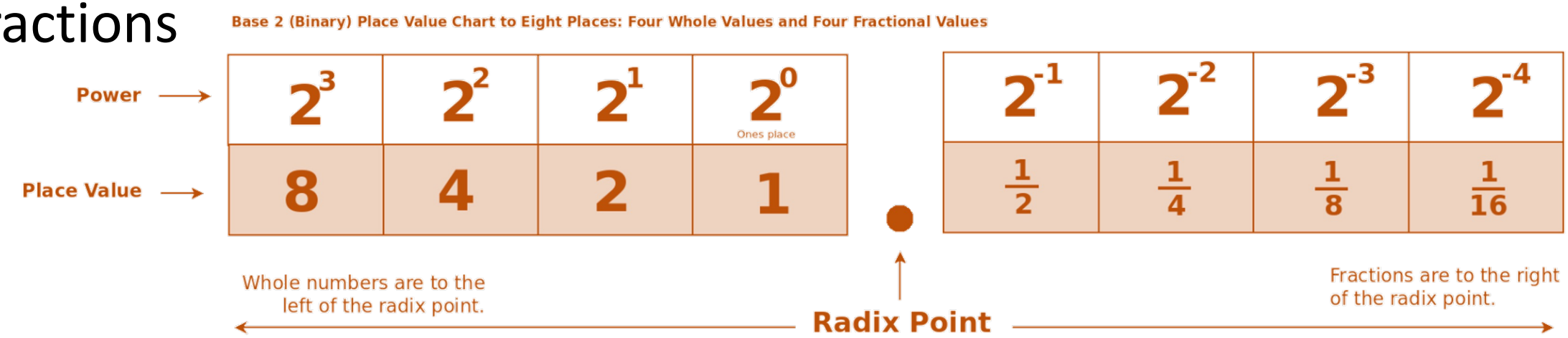
- Signed integer ("int")
  - One bit stores/represents sign (positive or negative)
  - Remaining bits store magnitude
  - Using 'n' bits can represent numbers from $-2^{n-1}+1,\ldots, 2^{n-1}-1$
- Unsigned integer ("unsigned int")
  - A non-negative integer
  - Using 'n' bits can represent numbers from $0,1,\ldots,2^{n}-1$

Number Representation Formats

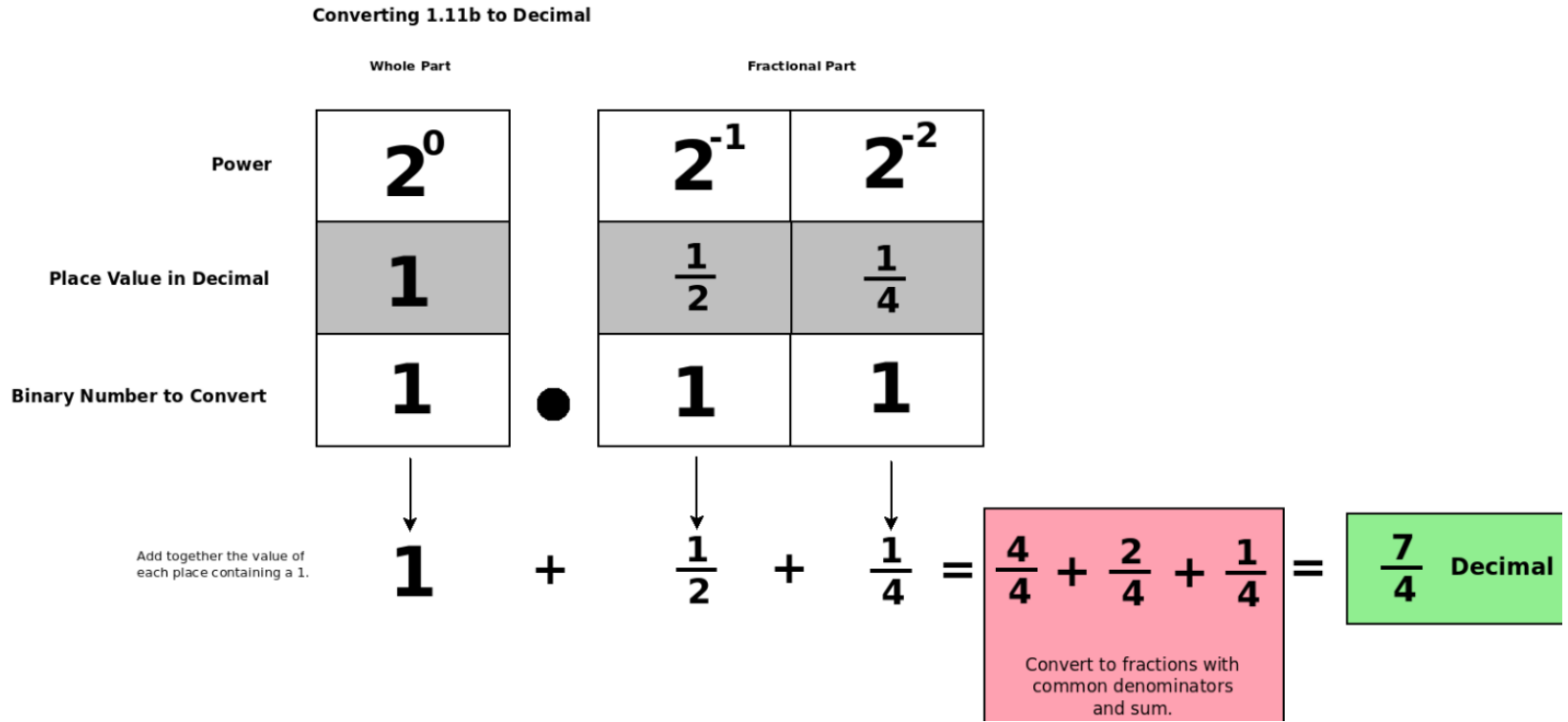
- "Floating-point" ("float")
  - Modeling real numbers approximately, up to finite precision
  - Uses 3 types of components (inspired from scientific notation)
    - Sign `-5.68434e-14` means $-5.68434 \times 10^{-14} = -0.0000000000000568434$
    - "Significand" = real number, usually having magnitude between 1 and 10
    - "Exponent" = integer
    - Example
      - Avogadro number in base 10: $6.02214076 \times 10^{23}\ \text{mol}^{-1}$
      - Avogadro number in base 2: $1.11111110001010101111111 \times 2^{1001110}$
      - Store sign and magnitude of significand
        - Decimal point can always be assumed after the first bit of the significand
      - Store sign and magnitude of exponent
    - Special representations
      - Infinity (+ and -)
      - NaN = Not-a-Number, resulting from operations like 0/0 or sqrt(-1) or $0 \times \infty$
    - Details in "IEEE 754" floating-point format

Number Representation Formats

- Binary fractions
  - Place values

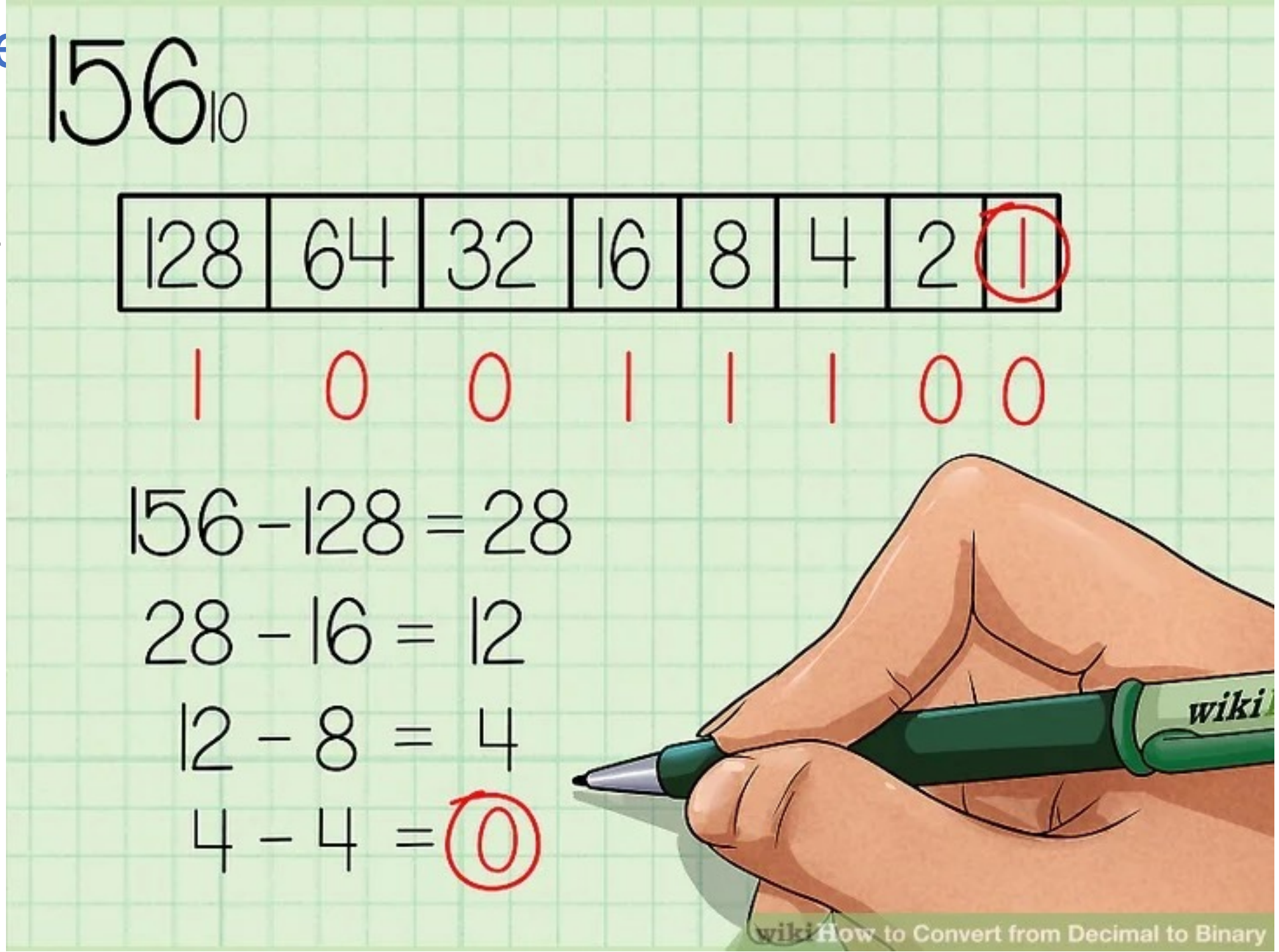


  - Convert 1.11 binary to decimal



Number Repre

- Binary fractions
  - Convert decimal integer to binary: **Method 1**
  - www.wikihow.com/Convert-from-Decimal-to-Binary



Number Represe

- Binary fractions
  - Convert decimal integer to binary: **Method 2**
  - www.wikihow.com/Convert-from-Decimal-to-Binary
  - e.g., 9_{10}

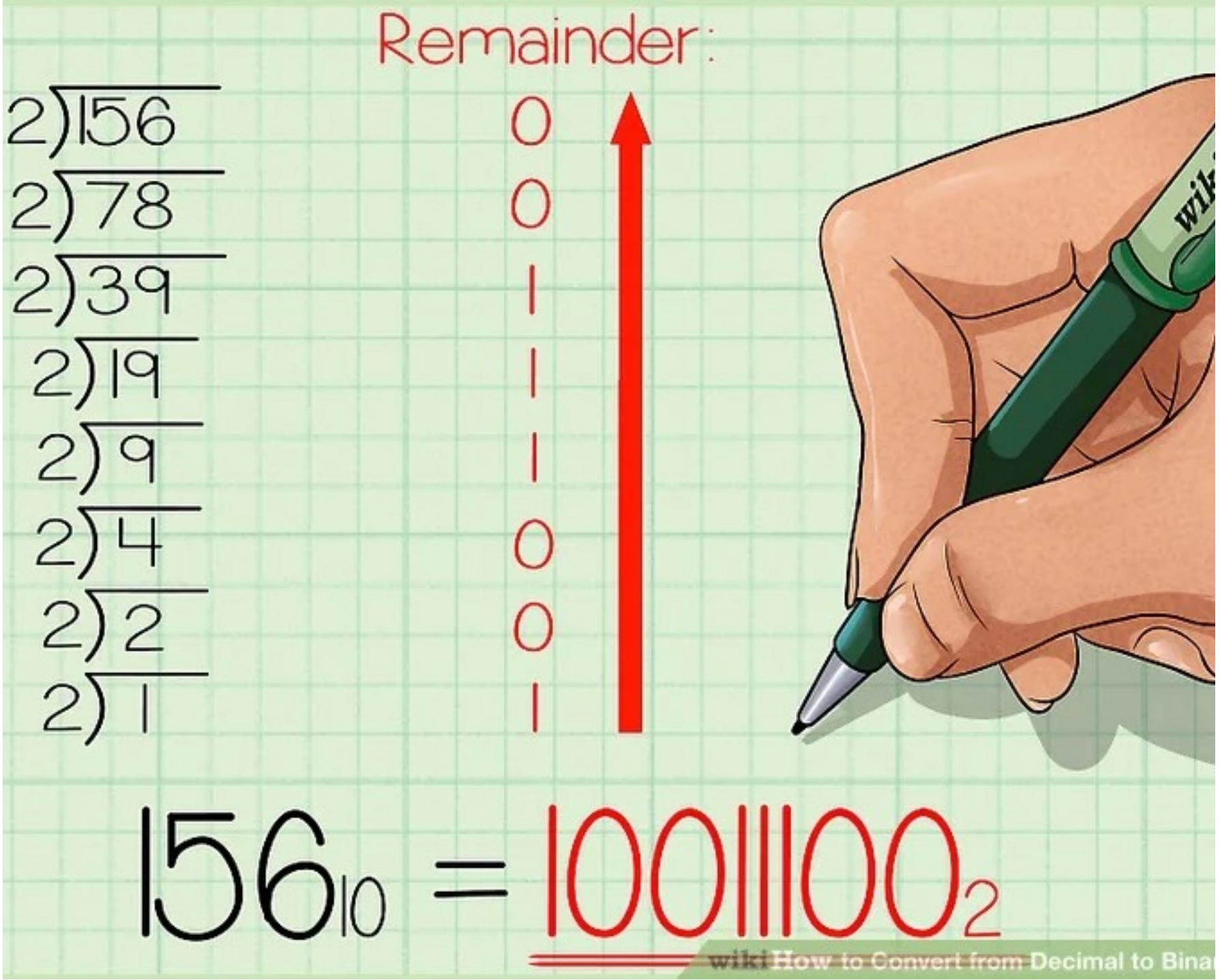
$= 2(4) + 1$

$= 2(2[2] + 0) + 1$

$= 2(2[2(1)+0] + 0) + 1$

$= 8*1 + 4*0 + 2*0 + 1$

$= 1001_2$



Number Representation Formats

- Binary fractions
  - Convert decimal to binary: **Method 3**
  - www.wikihow.com/Convert-from-Decimal-to-Binary
  - Convery 1.22 decimal to binary
  - If you want to compute upto 8 places after radix point, then start with round(1.22 * 2^8) = round(312.32) = 312
  - www.rapidtables.com/convert/number/decimal-to-binary.html?x=1.22

(312)/2	156	0	0
(156)/2	78	0	1
(78)/2	39	0	2
(39)/2	19	1	3
(19)/2	9	1	4
(9)/2	4	1	5
(4)/2	2	0	6
(2)/2	1	0	7
(1)/2	0	1	8

$= (100111000)_2 / 2^8$

$= (1.00111000)_2$

Number Representation Formats

- Floating-point
  - How is 123.45 stored in memory ?
  1. First converted to binary form: $123.45 = 1111011.01110011_2$
  2. The point is "floated" so that all non-zero bits are to the right

     $$123.45 = 0.11110110111001 1_2 \times 2^7$$

  3. For 32-bit float
     - Store sign (1 bit)
     - Store significand (23 bits) = `11110110111001 1`
     - Store exponent (8 bits) = 00000111

IEEE

- Institute of Electrical and Electronics Engineers
- Pronounced: I-triple-E
- Professional association for electronics engineering, electrical engineering, and other related disciplines
- Formed in 1963
- “Claims to produce over 30% of the world's literature in the electrical, electronics, and computer engineering fields”
  - 200 peer-reviewed journals and magazines
  - 1200 conference proceedings every year
- en.wikipedia.org/wiki/Institute_of_Electrical_and_Electronics_Engineers

IEEE

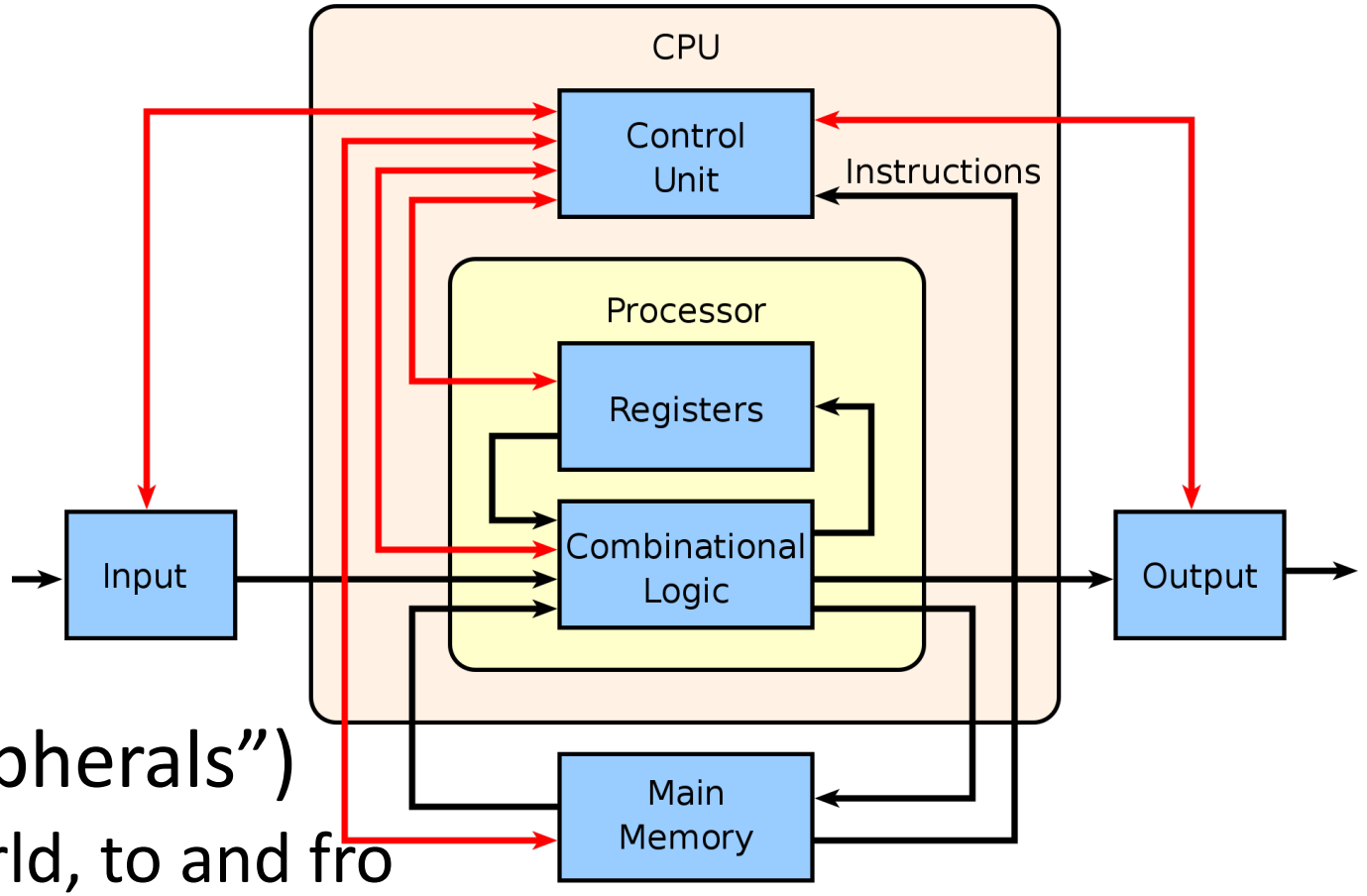
- Fields
  - IEEE Aerospace and Electronic Systems Society
  - IEEE Antennas & Propagation Society
  - IEEE Broadcast Technology Society
  - IEEE Circuits and Systems Society
  - IEEE Communications Society
  - IEEE Electronics Packaging Society
  - IEEE Computational Intelligence Society
  - IEEE Computer Society
  - IEEE Consumer Technology Society
  - IEEE Control Systems Society
  - IEEE Dielectrics & Electrical Insulation Society
  - IEEE Education Society
  - IEEE Electromagnetic Compatibility Society
  - IEEE Electron Devices Society
  - IEEE Engineering in Medicine and Biology Society
  - IEEE Geoscience and Remote Sensing Society
  - IEEE Industrial Electronics Society
  - IEEE Industry Applications Society
  - IEEE Information Theory Society
  - IEEE Instrumentation & Measurement Society
  - IEEE Intelligent Transportation Systems Society
  - IEEE Magnetics Society
  - IEEE Microwave Theory and Techniques Society
  - IEEE Nuclear and Plasma Sciences Society
  - IEEE Oceanic Engineering Society
  - IEEE Photonics Society
  - IEEE Power Electronics Society
  - IEEE Power & Energy Society
  - IEEE Product Safety Engineering Society
  - IEEE Professional Communication Society
  - IEEE Reliability Society
  - IEEE Robotics and Automation Society
  - IEEE Signal Processing Society
  - IEEE Society on Social Implications of Technology
  - IEEE Solid-State Circuits Society
  - IEEE Systems, Man, and Cybernetics Society
  - IEEE Technology and Engineering Management Society
  - IEEE Ultrasonics, Ferroelectrics, and Frequency Control Society
  - IEEE Vehicular Technology Society

Number Representation Formats

- NaN = Not-a-Number
- What is NaaN ?

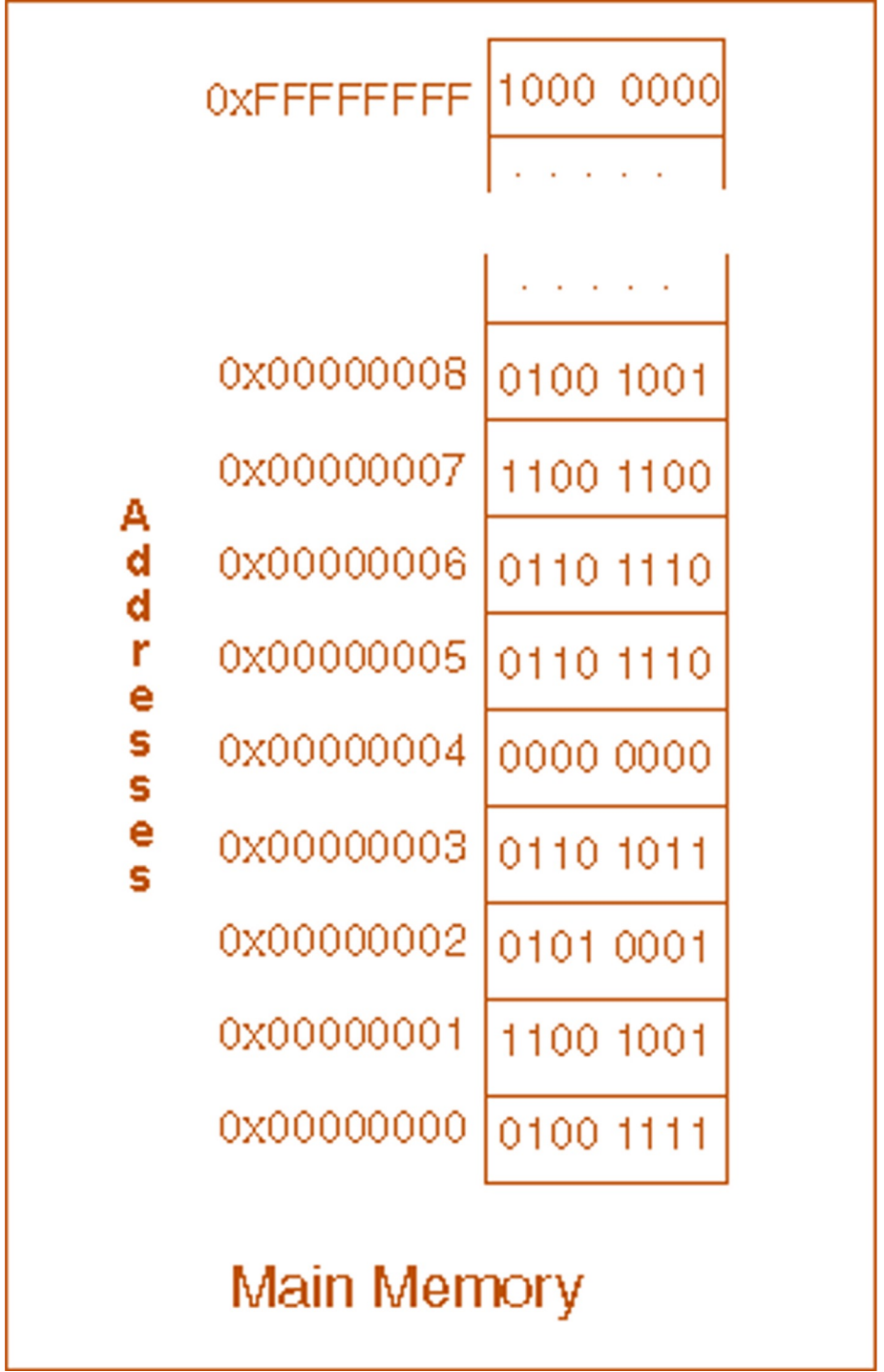
Parts of a Computer

- Main memory
  - Stores numbers/data, program/instructions
- Arithmetic (and logic) unit (ALU)
  - Performs arithmetic and logical operations on data
- Input-Output (IO) devices (“peripherals”)
  - To communicate with outside world, to and fro
    - e.g., keyboard, mouse, display screen, printer, disk
- Control unit: controls all units
- Central Processing Unit (CPU) = ALU + control unit
- Network: medium of transferring data between units



Main Memory

- Each byte of storage has an "address" / "location"
  - Byte = group of 8 bits
  - Memory with N bytes of storage has addresses going from 0 to 2^N-1
  - Addresses are also represented in binary (or hexadecimal = base-16)
- Example
  - CPU can execute instruction to store/"write" number N at address A
  - CPU can execute instruction to "read" number stored at address B



Control Unit

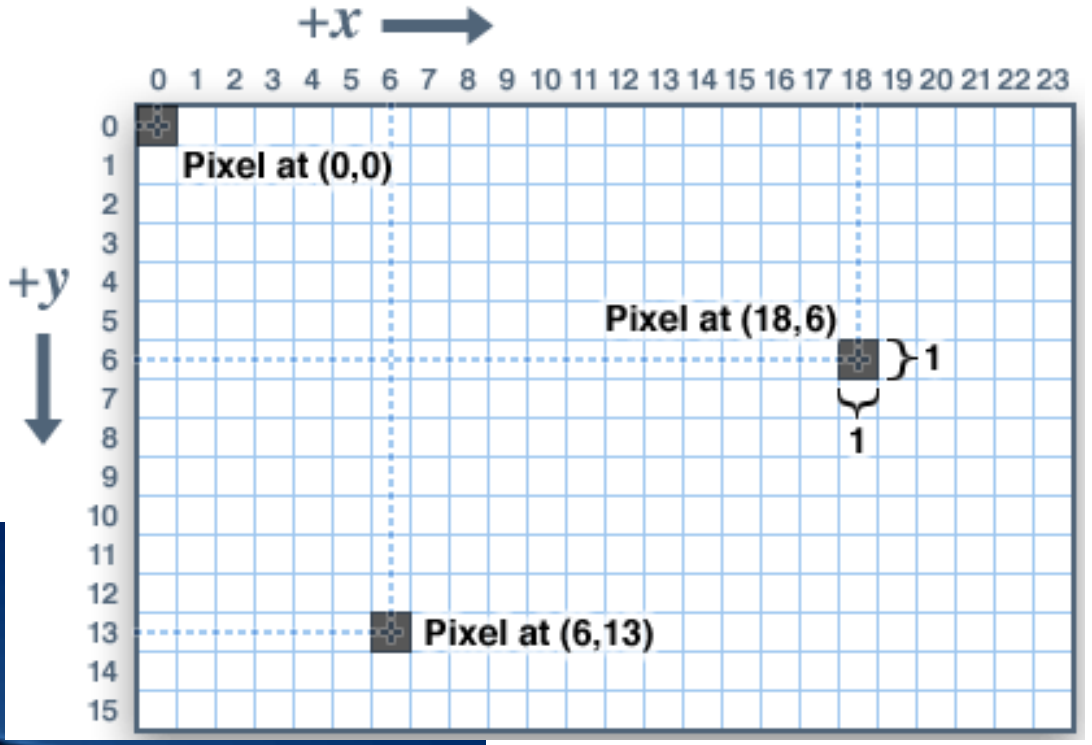
- Executes a sequence of operations (also represented as numbers)
  - "Machine language" used to communicate/specify sequence of instructions
  - e.g., a sequence of instructions (or a program) could be:
    53 100 57 100 100 104 57 100 104 104 63 104 99
    - 53, 100. This would read in a real number from the keyboard.
    - 57, 100, 100, 104. This would multiply the number in memory locations 100 with itself and put the result in location 104. Thus the square of the number will get placed in location 104. Note that after this instruction executes we will have the original number and its square in locations 100 and 104 respectively.
    - 57, 100, 104, 104. This would multiply the number in memory locations 100 and 104 and put the result in location 104. Thus the number and its square would get multiplied, and the result, the cube, would get placed in location 104.
    - 63,104. This would print out the cube on the screen.
    - 99. The program would stop.

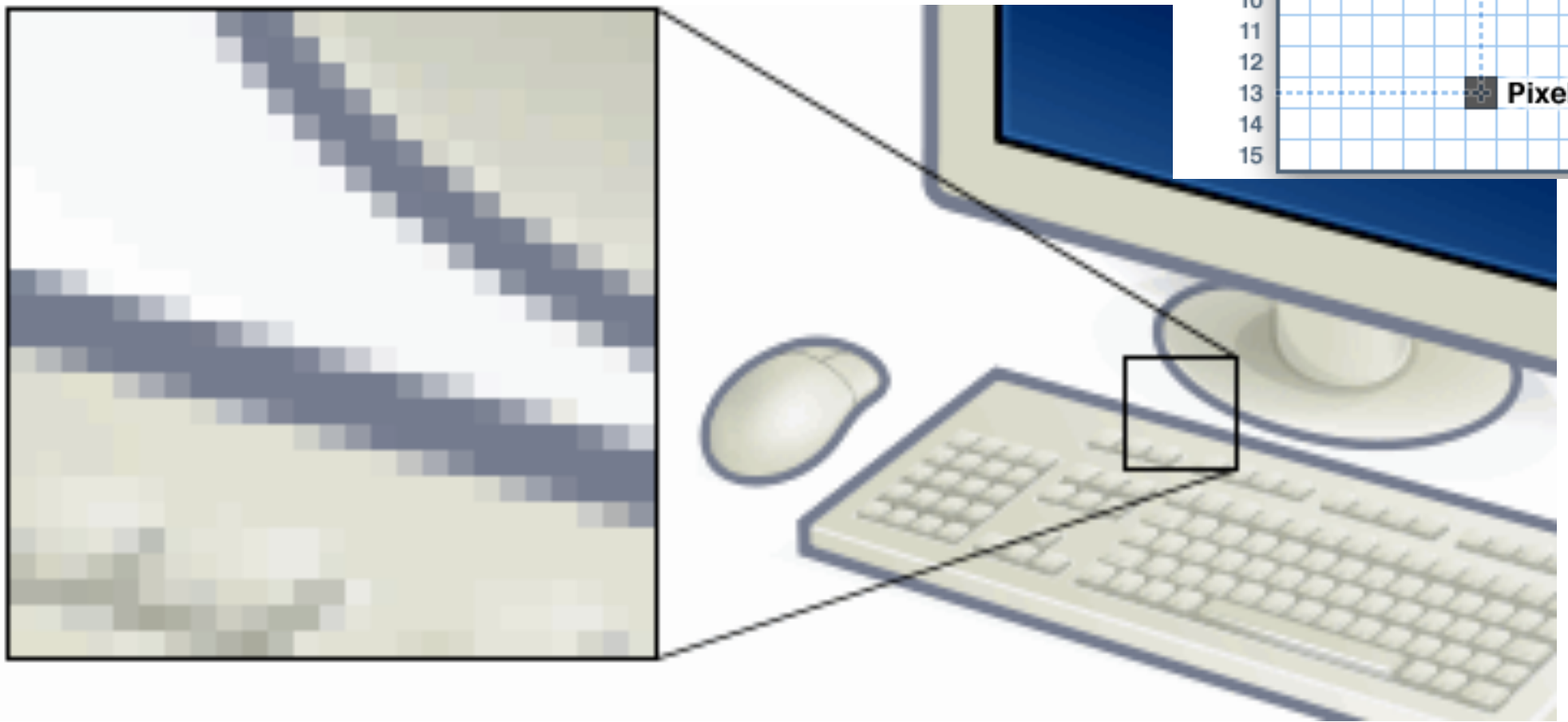
IO device: Keyboard

- Pressing each key (e.g., alphabetic character, numeral) sends a code to the computer

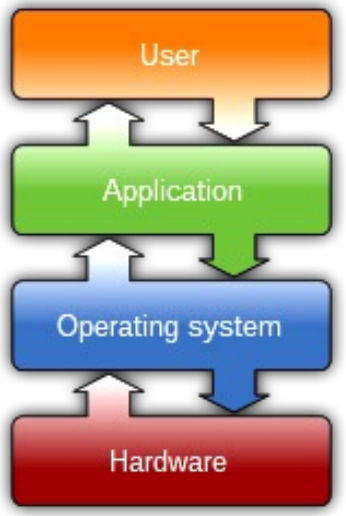
IO device: Display

- Pixels (picture elements) arranged in a grid
  - Pixel has a location
  - Pixel has a color





Operating System



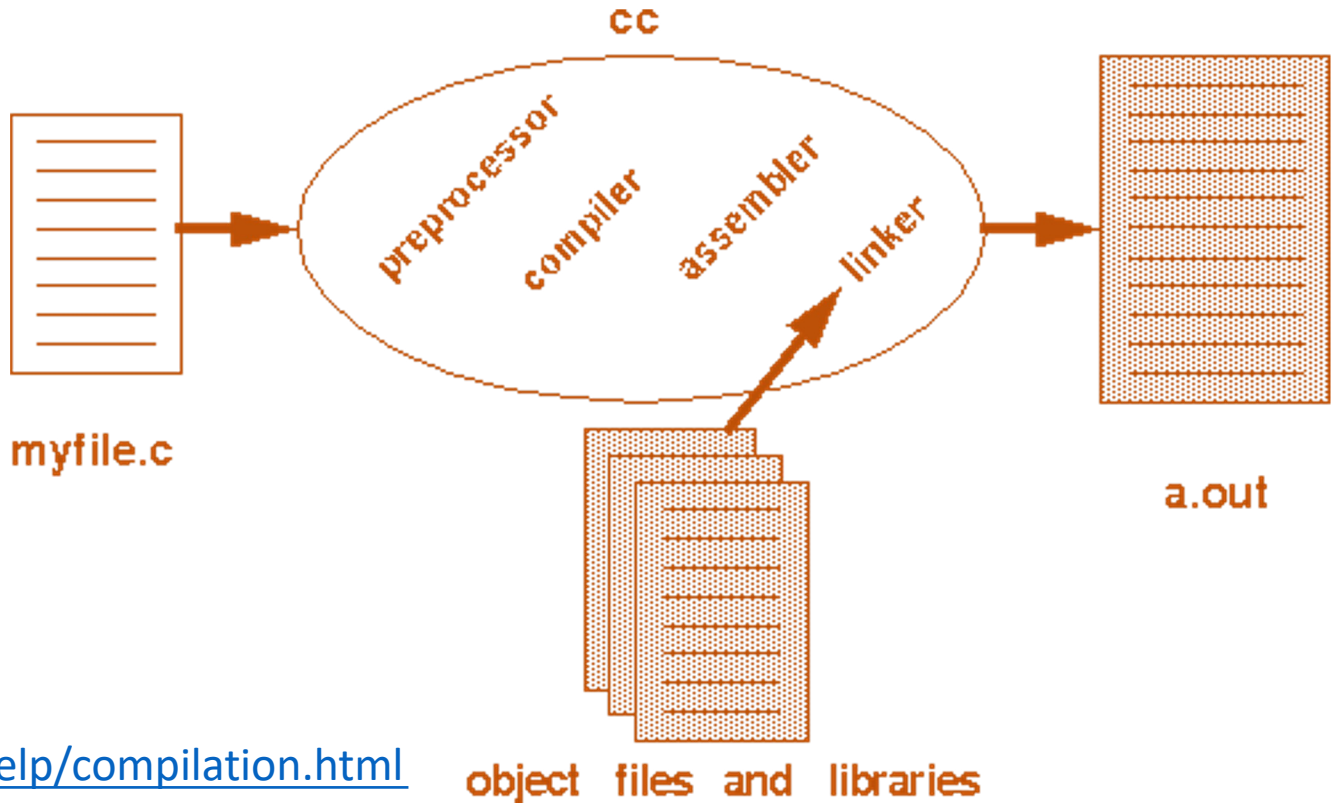
- Operating system (OS)
  - “Software that manages computer hardware and software resources, and provides common services for computer programs.”
    - en.wikipedia.org/wiki/Operating_system
  - UNIX: “a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix, whose development started in 1969 at Bell Labs research center by Ken Thompson, Dennis Ritchie, and others”
    - en.wikipedia.org/wiki/Unix
    - Unix was written in C and lower-level languages
  - Linux: “a family of open-source Unix-like operating systems based on Linux kernel, an operating system kernel first released in 1991, by Linus Torvalds”
    - en.wikipedia.org/wiki/Linux
    - Often packaged with other software provided by GNU project
    - GNU is a collection of free software; can be used as/with an OS
    - GNU is a recursive acronym: “GNU's Not Unix!”

Operating System

- Gnu: antelope found in African plains; Wildebeest

Programming Languages

- **Machine language** communicates directly with ALU
- We'd like higher-level abstractions for communication
  - Higher-level languages, e.g., simpleCPP
  - Program s++ contains softwares including a "compiler" that translates our instructions in simpleCPP language to a lower-level language and stores that in "a.out"
    - a.out = "assembler output"
- Translation software
  - cc: C Compiler
  - gcc: GNU C compiler
  - c++: C++ compiler
  - g++: GNU C++ compiler
  - s++: built using g++



courses.cs.washington.edu/courses/cse378/97au/help/compilation.html

Variables and Data Types

- "Variable" = region of memory, associated with a name, that is allocated for holding a data entity of a certain type/kind
  - A variable has a **name.** A variable has a memory **address.** A variable has a **type**
  - Variable's name can be used to refer to (or access) information/data at memory address
    - We refer to this information as the value of the variable
- Examples
  - General syntax to create/define a variable:

    ```
    data-type variable-name;
    ```

  - Example: `int nsides;`
  - Can create/define several variables in a single statement in a program

    ```
    data-type variable-name1, variable-name2, ... variable-namek;
    ```

Variables and Data Types

- Fundamental data types of C++

Data type	Possible values (Indicative)	# Bytes Allocated (Indicative)	Use for storing
`signed char`	-128 to 127	1	Characters or small integers.
`unsigned char`	0 to 255		
`short int`	-32768 to 32767	2	Medium size integers.
`unsigned short int`	0 to 65535		
`int`	-2147483648 to 2147483647	4	Standard size integers.
`unsigned int`	0 to 4294967295		
`long int`	-2147483648 to 2147483647	4	Storing longer integers.
`unsigned long int`	0 to 4294967295		
`long long int`	−9223372036854775808 to 9223372036854775807	8	Even longer integers.
`unsigned long long int`	0 to 18446744073709551615		
`bool`	`false` (0) or `true` (1)	1	Logical values.

Variables and Data Types

- How does signed char store numbers from **-128** to 127 ?
  - Table shows an example for a 3-bit storage system
    - Leftmost bit is indicative of sign
    - When sign is negative,
      get magnitude by
      flipping other bits and then adding 1
  - Twos-complement notation:
    en.wikipedia.org/wiki/
    two%27s_complement
    - Details are beyond scope of CS-101

Three-bit integers

Bits	Unsigned value	Signed value (Two's complement)
000	0	0
001	1	1
010	2	2
011	3	3
100	4	–4
101	5	–3
110	6	–2
111	7	–1

Variables and Data Types

- Fundamental data types of C++

`float`	Positive or negative. About 7 digits of precision. Magnitude in the range 1.17549×10^{-38} to 3.4028×10^{38}	4	Real numbers.
`double`	Positive or negative. About 15 digits of precision. Magnitude in the range 2.22507×10^{-308} to 1.7977×10^{308}	8	High precision and high range real numbers.
`long double`	Positive or negative. About 18 digits of precision. Magnitude in the range 3.3621×10^{-4932} to 1.18973×10^{4932}	12	High precision and very high range real numbers.

Variables and Data Types

- Fundamental data types of C++
  - Storage allocated for different data types might vary across machines
  - Use sizeof() operator to find out exactly how many bytes would be allocated

```
cout << "\t          char: " <<          sizeof(char) << endl;
cout << "\t         short: " <<         sizeof(short) << endl;
cout << "\t           int: " <<           sizeof(int) << endl;
cout << "\t          long: " <<          sizeof(long) << endl;
cout << "\t unsigned char: " <<  sizeof(unsigned char) << endl;
cout << "\tunsigned short: " << sizeof(unsigned short) << endl;
cout << "\t  unsigned int: " <<   sizeof(unsigned int) << endl;
cout << "\t unsigned long: " <<  sizeof(unsigned long) << endl;
cout << "\t   signed char: " <<    sizeof(signed char) << endl;
cout << "\t         float: " <<          sizeof(float) << endl;
cout << "\t        double: " <<         sizeof(double) << endl;
cout << "\t   long double: " <<    sizeof(long double) << endl;
```

```
          char: 1
         short: 2
           int: 4
          long: 4
 unsigned char: 1
unsigned short: 2
  unsigned int: 4
 unsigned long: 4
   signed char: 1
         float: 4
        double: 8
   long double: 8
```

Variable Naming

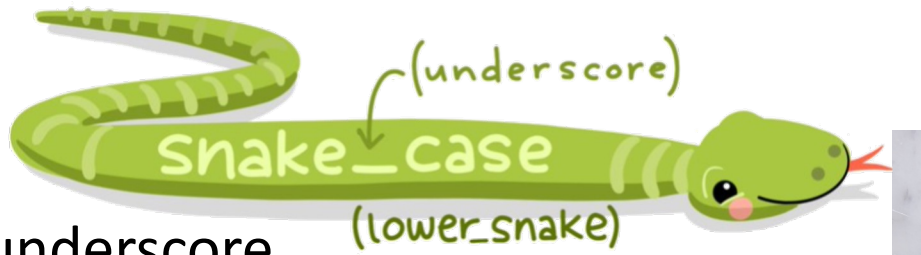
- Design name to indicate the intended purpose of variable in program
  - E.g., “int mm” creates/defines a variable called “mm”, but where the name “mm” seems too cryptic
  - E.g., “int mathMarks” is a better statement
- Design data-type according to intended values of variable
  - E.g., “unsigned int numSides”, or “unsigned char numSides”, or “unsigned int mathMarks”
- Variable names are case sensitive
  - E.g., mathMarks, MathMarks, mathmarks, are all different variables
- Create/define variable when and where you need it in the program
  - Creating variable in line 10 and using it first on line 500 isn’t helpful to reader

Variable Naming

- Variable naming convention
  - en.wikipedia.org/wiki/Naming_convention_(programming)
  - “To reduce the effort needed to read and understand source code”
  - “To enable code reviews to focus on issues more important than syntax”
  - Example
    - Compare `a = b * c;` vs. `weekly_pay = hours_worked * hourly_pay_rate;`
      - Note: spaces aren’t allowed in variable names; underscores are allowed
  - Length of name; tradeoffs
    - Shorter names can be easier to type (but some editing software allow text completion), but can be too cryptic
    - Longer names can give more insight into purpose and use of variable, but very long names lead to visual clutter

Variable Naming

- Character “case”
  - Uppercase: e.g., PI
  - Lowercase: e.g., count
  - Camel case
    - Without spaces. With capitalized words.
    - e.g., wordCount, iPhone, eBay
    - e.g., YouTube
  - Snake case
    - Replace space by underscore
    - e.g., word_count
  - Kebab case (doesn’t work with C++)
    - Replace space by underscore
    - e.g., the-quick-brown-fox-jumps-over-the-lazy-dog



Variable Initialization

- Creating/defining a variable "int numSides"
  - Allocates memory region of size 4 bytes
  - Associated name 'numSides" with that memory location
  - Doesn't specify any value of variable, or content in memory region
    - By default, this memory can contain any junk information
- Good idea to initialize variable values, when it makes sense
  - E.g., "unsigned int numOfSides = 0"
- What does this statement do ? "char letter_a = 'a'; "
  - Stores number 97 (code for character 'a') in letter_a
  - ASCII en.wikipedia.org/wiki/ASCII is a standard for character encoding in electronic communication
    - American Standard Code for Information Interchange

ASCII

- codes

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

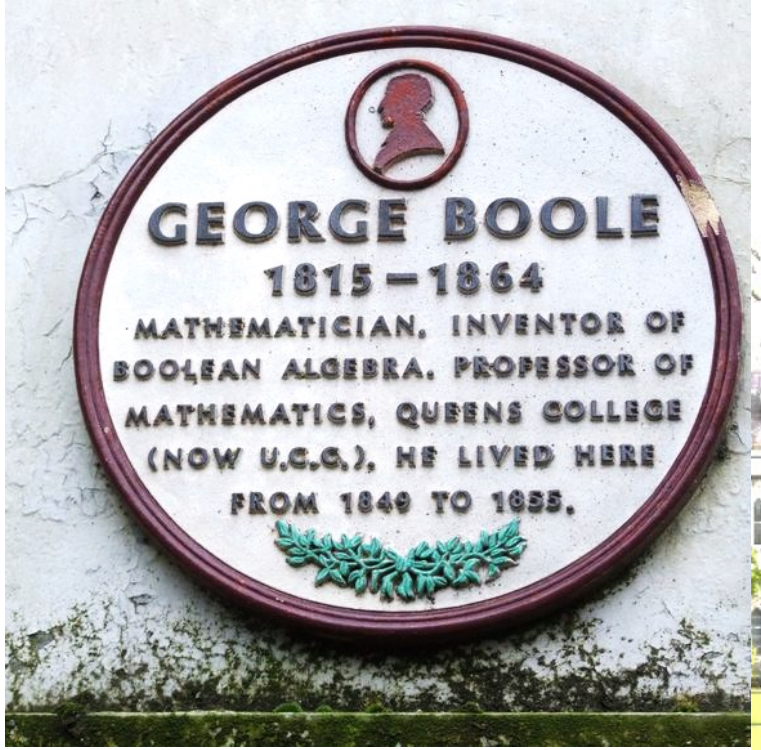
ASCII codes

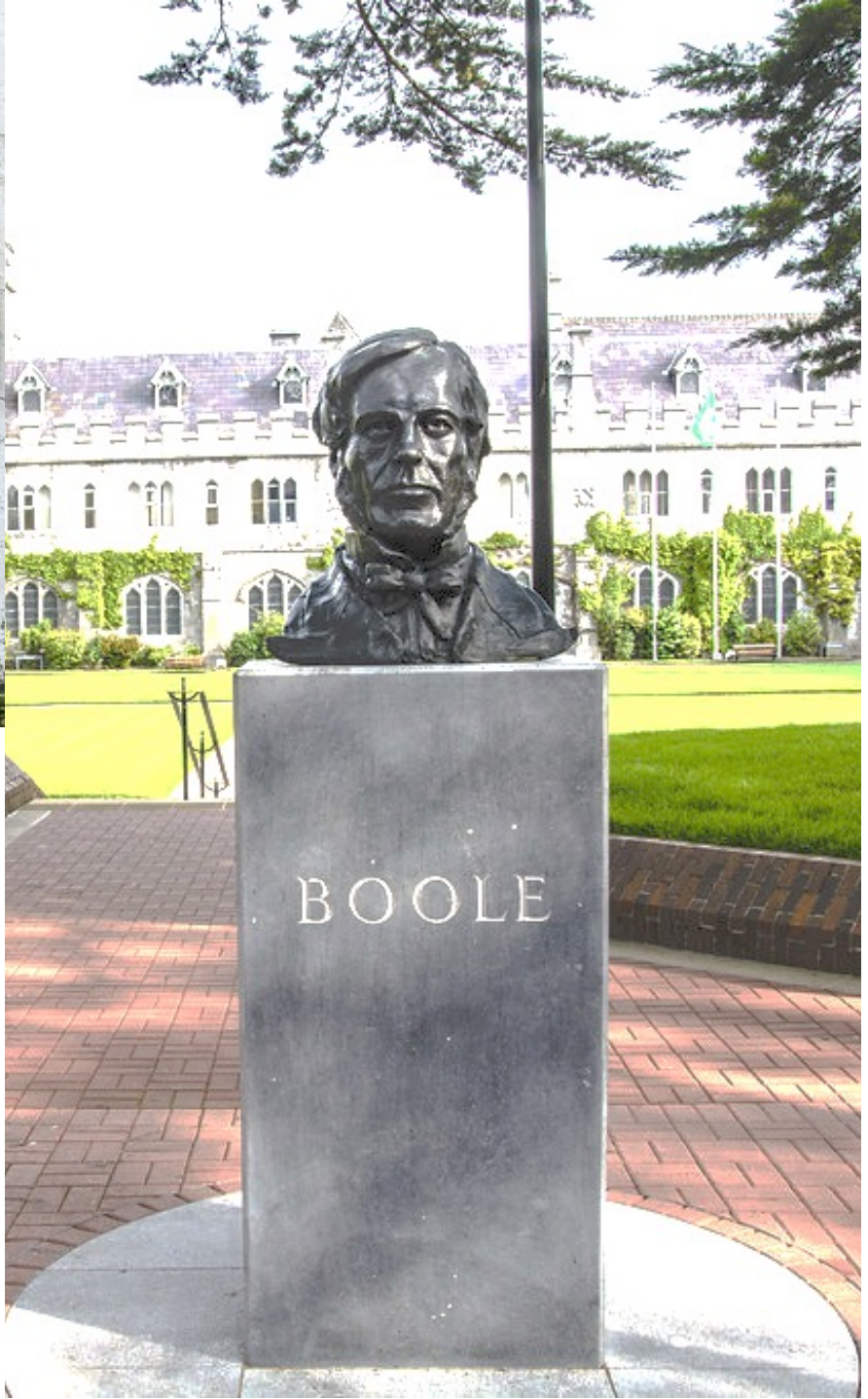
- What will this type ?

```
main_program{
  char small, capital;

  cout << "Type in any lower case letter: ";
  cin >> small;

  capital = small + 'A' - 'a';

  cout << capital << endl;
}
```

Boolean

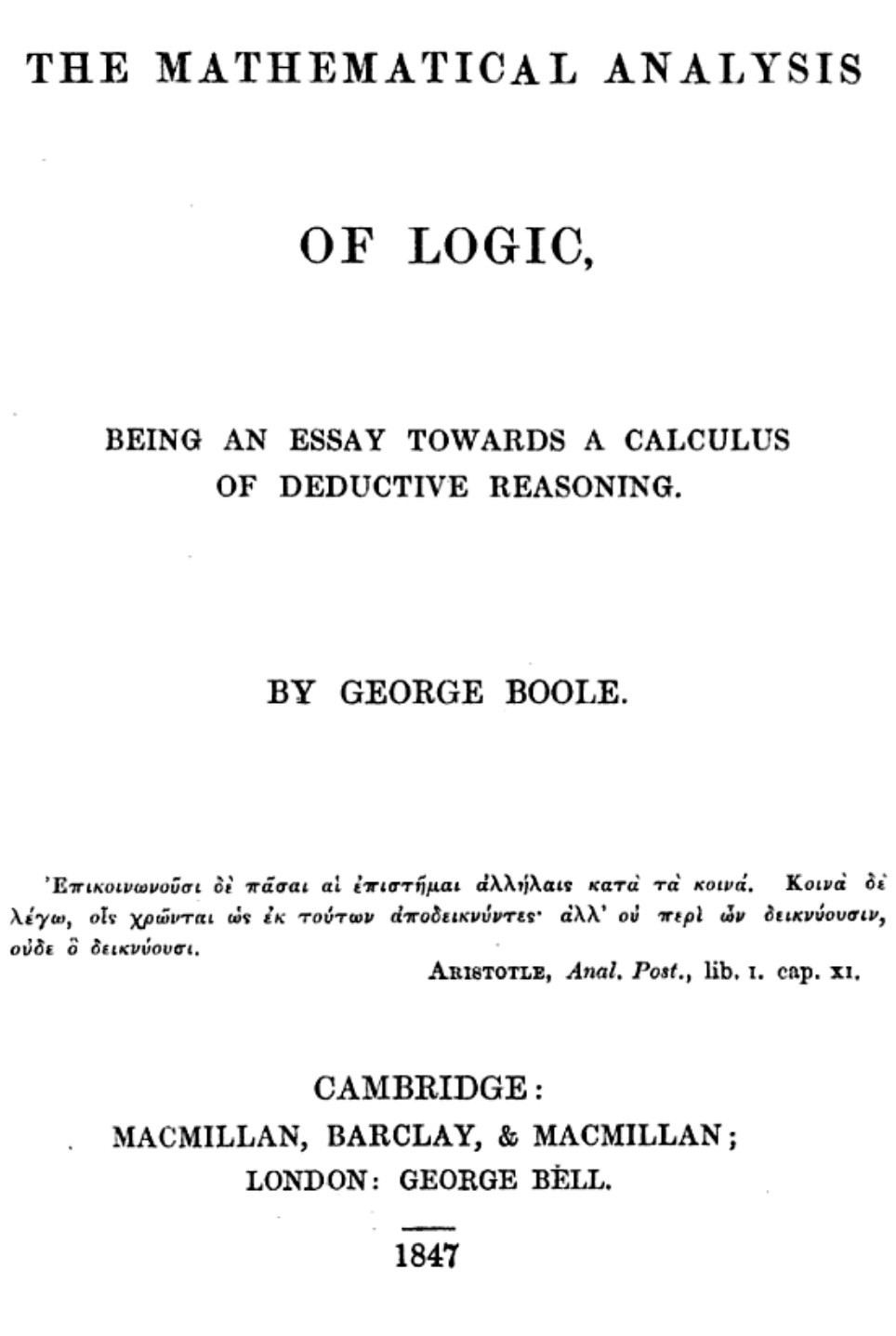
- Boolean data type,
  e.g.,
  “bool penDown = true”
- George Boole
  - Self-taught
    English mathematician,
    philosopher, and logician
  - Son of a shoemaker
  - At 16, he began teaching to support his family
  - At 19, he established his own school
  - First professor of mathematics
    at University College Cork





Boolean

- George Boole
  - In 1847 (age 32), Boole developed Boolean algebra, a fundamental concept in binary logic



THE MATHEMATICAL ANALYSIS

OF LOGIC,

BEING AN ESSAY TOWARDS A CALCULUS
OF DEDUCTIVE REASONING.

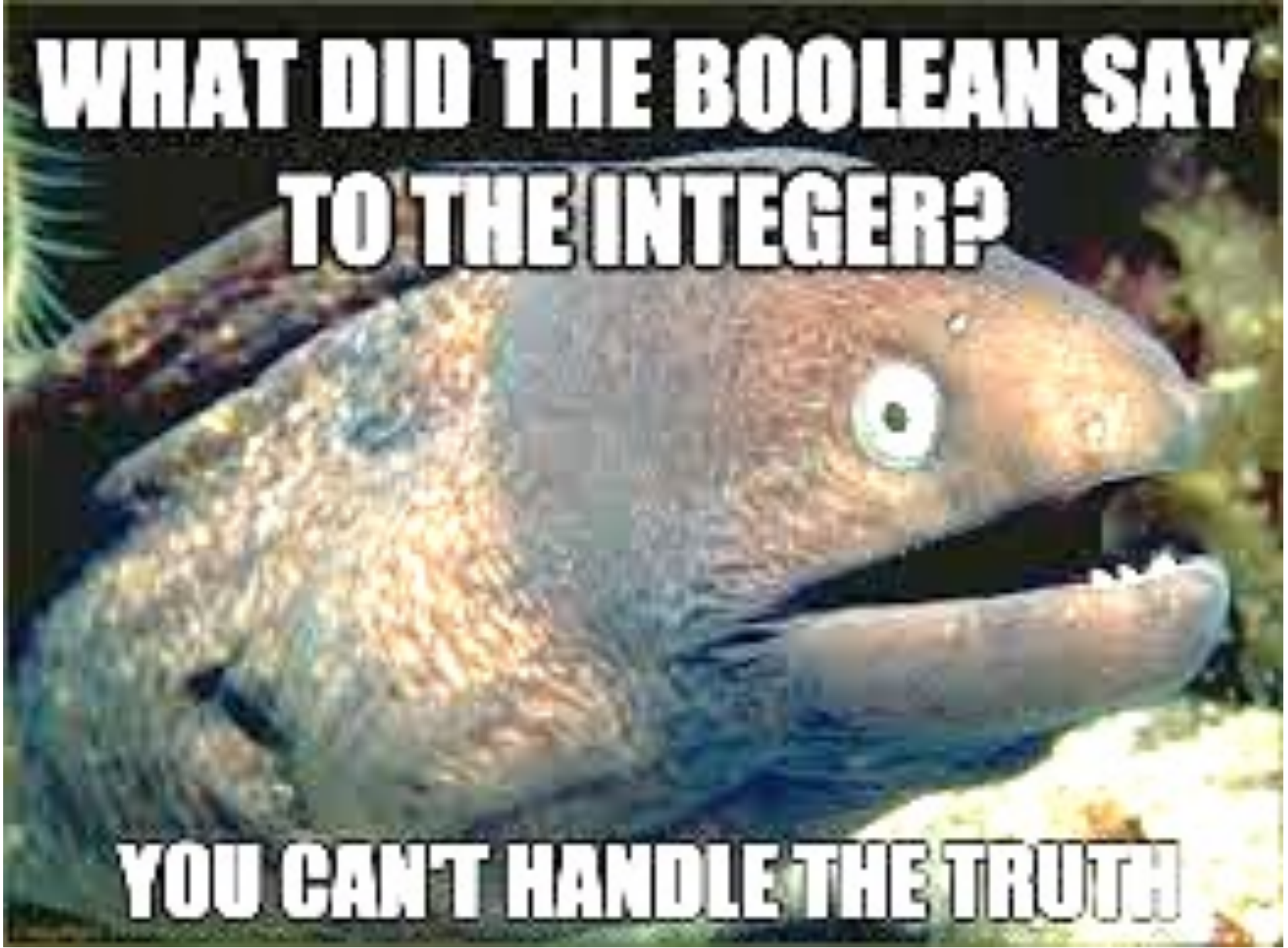
BY GEORGE BOOLE.

Ἐπικοινωνοῦσι δὲ πᾶσαι αἱ ἐπιστῆμαι ἀλλήλαις κατὰ τὰ κοινά. Κοινὰ δὲ λέγω, οἷς χρῶνται ὡς ἐκ τούτων ἀποδεικνύντες· ἀλλ' οὐ περὶ ὧν δεικνύουσιν, οὐδὲ ὃ δεικνύουσι.

ARISTOTLE, *Anal. Post.*, lib. I. cap. XI.

CAMBRIDGE:
MACMILLAN, BARCLAY, & MACMILLAN;
LONDON: GEORGE BELL.

1847


WHAT DID THE BOOLEAN SAY TO THE INTEGER?
YOU CAN'T HANDLE THE TRUTH

In Lighter Vein: https://bhailang.js.org

General

`hi bhai` is the entrypoint for the program and all program must end with `bye bhai`. Anything outside of it will be ignored.

```
This will be ignored

hi bhai
  // Write code here
bye bhai

This too
```

Variables

Variables can be declared using `bhai ye hai`.

```
hi bhai
    bhai ye hai a = 10;
    bhai ye hai b = "two";
    bhai ye hai c = 15;
    a = a + 1;
    b = 21;
    c *= 2;
bye bhai
```

Built-ins

Use `bol bhai` to print anything to console.

```
hi bhai
    bol bhai "Hello World";
    bhai ye hai a = 10;
    {
       bhai ye hai b = 20;
       bol bhai a + b;
    }
    bol bhai 5, 'ok', nalla , sahi , galat;
bye bhai
```

Types

Numbers and strings are like other languages. Null values can be denoted using `nalla`. `sahi` and `galat` are the boolean values.

```
hi bhai
    bhai ye hai a = 10;
    bhai ye hai b = 10 + (15*20);
    bhai ye hai c = "two";
    bhai ye hai d = 'ok';
    bhai ye hai e = nalla;
    bhai ye hai f = sahi;
    bhai ye hai g = galat;
bye bhai
```

Variable Initialization

- Variables whose values stay unchanged
  - Use the “const” keyword
  - e.g., `const float Avogardro = 6.022E23;`
  - If your program tries to change it later, then it will result in an error
  - Semantics of the “const” keyword is that the value of a variable is to be fixed when the variable is created/instantiated

Reading Data into a Variable

- Statement: cin >> varName;
  - When statement is executed, program waits to receive input from keyboard
    - End of input typically indicated by keying “enter”/newline
  - Initial “whitespace” characters will be ignored before we type in a value consistent with the data type of varName
    - Whitespace characters = those representing horizontal or vertical space in typography
    - These are invisible, but occupy space on a page, e.g., space (‘ ’), tab (‘\t’), newline (‘\n’)
    - Case 1: varName is of numeric type
      - Initial whitespaces ignored
    - Case 2: varName is of char type
      - Initial whitespaces ignored
      - ASCII value of first non-whitespace character is assigned to varName
- Reading many values: cin >> varName1 >> varName2;

Printing Variable Values on Display

- Statement: cout << varName;
- If you want a end-of-line to be printed,
  use:
  cout << varName << endl;
  or use:
  cout << varName << "\n";
- Consider: char c = 97; cout << c << endl;
  - This will print the character whose ASCII value is 97
- Printing several values
  - e.g., cout << varName1 << endl << varName2 << endl;
  - e.g., cout << "varName1: " << varName1 << ", "
    << "varName2: " << varName2 << endl;

Assignment

- Syntax: variable = expression;
- Example

```
int x=2,y=3,p=4,q=5,r;
r = x*y + p*q;
cout << x*y+p*q << endl;
```

- Example
  - Valid statement: p = p + 1;
  - First evaluate expression on right-hand-side;
    Then assign expression value to variable on left
- Example
  - Invalid statement: p+1 = p;
  - Left hand side isn't a variable

Evaluating Expressions

- Precedence of operators
  - In mathematics and computer programming, the order of operations (or operator precedence) is a collection of rules that reflect conventions about which procedures to perform first in order to evaluate a given mathematical expression
    - e.g., $1 + 2 \times 3$ is interpreted to have the value $1 + (2 \times 3) = 7$, and not $(1 + 2) \times 3 = 9$
  - Order
    1. Parenthetic subexpressions
    2. Exponentiation
    3. Multiplication and Division
    4. Addition and Subtraction
  - More details:
    - naipc.uchicago.edu/2014/ref/cppreference/en/cpp/language/operator_precedence.html

Assignment

- The assignment statement is itself also an expression !
- Example

```
int x,y,z;
x = y = z = 1;
```

- Associativity of operator “=” is right-to-left
  - Rightmost “=” assignment is evaluated/executed first
  - Its expression value is 1, which is then used to assign “y”

```
x = (y = (z = 1));
```

Integer Division

- Consider

```
int m=100, n=7, p, q;
p = m/n;
q = 35/200;
```

  - Value of p will be 14. Value of q will be 0.
  - Integer division produces the integer part (floor) of the decimal quotient

- Example:

```
cout <<"Give the duration in seconds: ";
int duration; cin >> duration;
int hours, minutes, seconds;
hours = duration/3600;
minutes = (duration - hours*3600)/60;
seconds = duration % 60;
cout <<"Hours: "<< hours <<", Minutes: "
     << minutes <<", Seconds: "<< seconds << endl;
```

  - When input is 5000, output is: `Hours: 1, Minutes: 23, Seconds: 20`

Integer Division

- Example

```
int x=100, w;
float y,z;
y = 360/x;
z = 360.0/x;
w = 360.0/x;
```

  - Value of y will be ? 3
  - Value of z will be ? 3.6
  - Value of w will be ? 3

- Syntax for explicit type conversion:
  "type (expression)" or "(type) expression"
  - e.g., "y = float(360) / x" will lead to value of y being 3.6

Example

- Be careful about precedence rules, order of evaluation, integer division
- Example

```
main_program{
    double centigrade, fahrenheit;

    cout << "Give temperature in Centigrade: ";
    cin >> centigrade;

    fahrenheit = 32.0 + centigrade * 9.0/5.0;
    cout << "Temperature in Fahrenheit: " << fahrenheit << endl;
}
```

  - What if we wrote 9/5 instead of 9.0/5.0 ?
    - Will division happen first or will multiplication happen first ?
      - When multiplication happens, it will be done in double precision
    - Mixing floating-point and integer variables in this way leaves room for ambiguity. Avoid such code.

Assignment with Repeat

- What will this do ?

```
main_program{
  turtleSim();
  int i = 1;
  repeat(10){
    forward(i*10); right(90);
    forward(i*10); right(90);
    i = i + 1;
  }
  wait(5);
}
```

Increment and Decrement Operators

- Post-increment operator: ++ as a suffix
  - int a = 1, b;
    b = a++;
  - Use variable ‘a’ in evaluating expression, and increment variable after use
  - So, after execution: value of b is 1, value of a is 2
- Pre-increment operator: ++ as a prefix
  - int a = 1, b;
    b = ++a;
  - Increment variable, and then use incremented variable to evaluate expression
  - So, after execution: value of b is 2, value of a is 2
- Less-confusing and preferred way
  - Write b = a; a++; OR
  - Write a++; b = a;

Compound Assignment Operators

- Instead of writing
  varName = varName + expression;
  one could write
  varName += expression;
- Similar usage for -=, *=, /=
- Usually we avoid such usage,
  to prefer clarity and easy of understanding
  - Because we are usually more familiar with standard assignment operator “=”
  - Because these may be prone to misreading (e.g., as “=”)

Assignment with Repeat

- What will this do ?

```
main_program{
   int count;
   cout << "How many numbers: ";
   cin >> count;

   float num,sum=0;
   repeat(count){
      cout << "Give the next number: ";
      cin >> num;
      sum = sum + num;
   }

   cout << "Average is: ";
   cout << sum/count;
   cout << endl;
}
```

Block, Scope

- Block = region of program starting from "{" to the corresponding "}"
  - If we consider two blocks within a program, then
    they must be entirely disjoint OR
    one block must be entirely contained (nested) within the other
- Parent block of a variable definition =
  innermost block in which variable is defined
- "Scope" of a variable = region of program where the variable can be referenced/used, i.e., where the variable is "visible"
  - A variable is created when its definition is encountered during execution; variable is destroyed when its parent block is exited
  - Notion of scope helps prevent name collisions by allowing the same name to refer to different objects – as long as the names have separate scopes
    - Shadowing: Variable defined within a scope shadows variable defined in outer scope

Assignment with Repeat

- What will this do ?
  - A different way
  - Need to create and destroy variable num within each iteration of repeat loop, leading to extra overhead for bookkeeping
  - Prevent inadvertent access of variable num outside loop
    - So this is the preferred way because it is safer and more readable

```
main_program{
  int count;
  cout << "How many numbers: ";
  cin >> count;

  float sum=0;
  repeat(count){
    cout << "Give the next number: ";
    float num;
    cin >> num;
    sum = sum + num;
  }

  cout << "Average is: ";
  cout << sum/count;
  cout << endl;
}
```

Practice Examples for Lab: Set 2

- 1 Write a program that prints the arithmetic sequence $a, a+d, a+2d, \ldots, a+nd$. Take a, d, n as input.

  Done

  Write a program that prints out the geometric sequence $a, ar, ar^2, \ldots, ar^n$, taking a, r, n as input.

  Done

- 2 Write a program that reads in distance d in inches and prints it out as v miles, w furlongs, x yards, y feet, z inches. Remember that a mile equals 8 furlongs, a furlong equals 220 yards, a yard is 3 feet, and a foot is 12 inches. So your answer should satisfy $d = (((8v + w) \cdot 220 + x) \cdot 3 + y) \cdot 12 + z$, and further $w < 8, x < 220, y < 3, z < 12$.

  done

- 3 What is the value of `x` after the following statements are executed? (a) `x=22/7;` (b) `x=22.0/7;` (c) `x=6.022E23 + 1 - 6.022E23` (d) `x=6.022E23 - 6.022E23 + 1` (e) `x=6.022E23 * 6.022E23`. Answer for three cases, when `x` is defined to be of type `int`, `float`, `double`. Put these statements in a program, execute and check your conclusions.

Practice Examples for Lab: Set 2

- 4 What will be the effect of executing the following code fragment?

done

```
float f1, f2,centigrade=100;
f1 = centigrade*9/5 + 32;
f2 = 32 + 9/5*centigrade;
cout << f1 << ' ' << f2 << endl;

char x = 'a', y;
y = x + 1;
cout << y << ' ' << x + 1 << endl;
```

done

- 5 For what values of `a,b,c` will the expressions `a+(b+c)` and `(a+b)+c` evaluate to different values?

- 6 Draw a smooth spiral. The spiral should wind around itself in a parallel manner, i.e. there should be a certain point called "center" such that if you draw a line going out from it, the spiral should intersect it at equal distances as it winds around.

Numeric Overflow

- Integer “overflow”
  - Occurs when manipulating integers beyond their maximum allowed value

```
{ // prints n until it overflows:
  int n=1000;
  cout << "n = " << n << endl;
  n *= 1000;  // multiplies n by 1000
  cout << "n = " << n << endl;
  n *= 1000;  // multiplies n by 1000
  cout << "n = " << n << endl;
  n *= 1000;  // multiplies n by 1000
  cout << "n = " << n << endl;
}
```

```
n = 1000
n = 1000000
n = 1000000000
n = -727379968
```

Numeric Overflow

- Floating-point overflow
  - Occurs when manipulating floating-point numbers beyond their maximum allowed value

```
{ // prints x until it overflows:
  float x=1000.0;
  cout << "x = " << x << endl;
  x *= x;  // multiplies n by itself; i.e., it squares x
  cout << "x = " << x << endl;
  x *= x;  // multiplies n by itself; i.e., it squares x
  cout << "x = " << x << endl;
  x *= x;  // multiplies n by itself; i.e., it squares x
  cout << "x = " << x << endl;
  x *= x;  // multiplies n by itself; i.e., it squares x
  cout << "x = " << x << endl;
}
```

```
x = 1000
x = 1e+06
x = 1e+12
x = 1e+24
x = inf
```

Numeric Round-Off Error

- Floating-point round-off error

```
double x = 1000/3.0;cout << "x = " << x << endl;  // x = 1000/3
double y = x - 333.0;cout << "y = " << y << endl; // y = 1/3
double z = 3*y - 1.0;cout << "z = " << z << endl; // z = 3(1/3) - 1
```

```
x = 333.333
y = 0.333333
z = -5.68434e-14
```

E-Format for Floating-Point Values

- Floating-point
  - “cin” can also read in scientific format

```
{ // prints double values in scientific e-format:
  double x;
  cout << "Enter float: ";  cin >> x;
  cout << "Its reciprocal is: " << 1/x << endl;
}
Enter float: 234.567e89
Its reciprocal is: 4.26317e-92
```

Block, Scope

- Example
  - What will the output be ?

```
main_program{
  int sum=0;
  repeat(5){
    int num;                    // statement 1
    cin >> num;
    sum += num;
  }
  cout << sum << endl;
  int prod=1;
  repeat(5){
    int num;                    // statement 2
    cin >> num;
    prod *= num;
  }
  cout << prod << endl;
}
```

Block, Scope

- Example
  - What will the output be ?

10
5
10
5
10
5
10

```
main_program{
   int p=10;                    // statement 3
   repeat(3){
      cout << p << endl;        // statement 4
      int p=5;                  // statement 5
      cout << p << endl;        // statement 6
   }
   cout << p << endl;           // statement 7
}
```

  - Variable ‘p’ defined/created in statement 5 shadows variable ‘p’ defined/created in statement 3

Block, Scope

- Example
  - What will the output be ?
    - Trick question: compilation error

```
#include<simplecpp>

main_program {

  int p = 10;
  repeat (3)
    {
      cout << p << a << endl;
      int a = 5;
      cout << p << a << endl;
    }
  cout << p << endl;
}
```

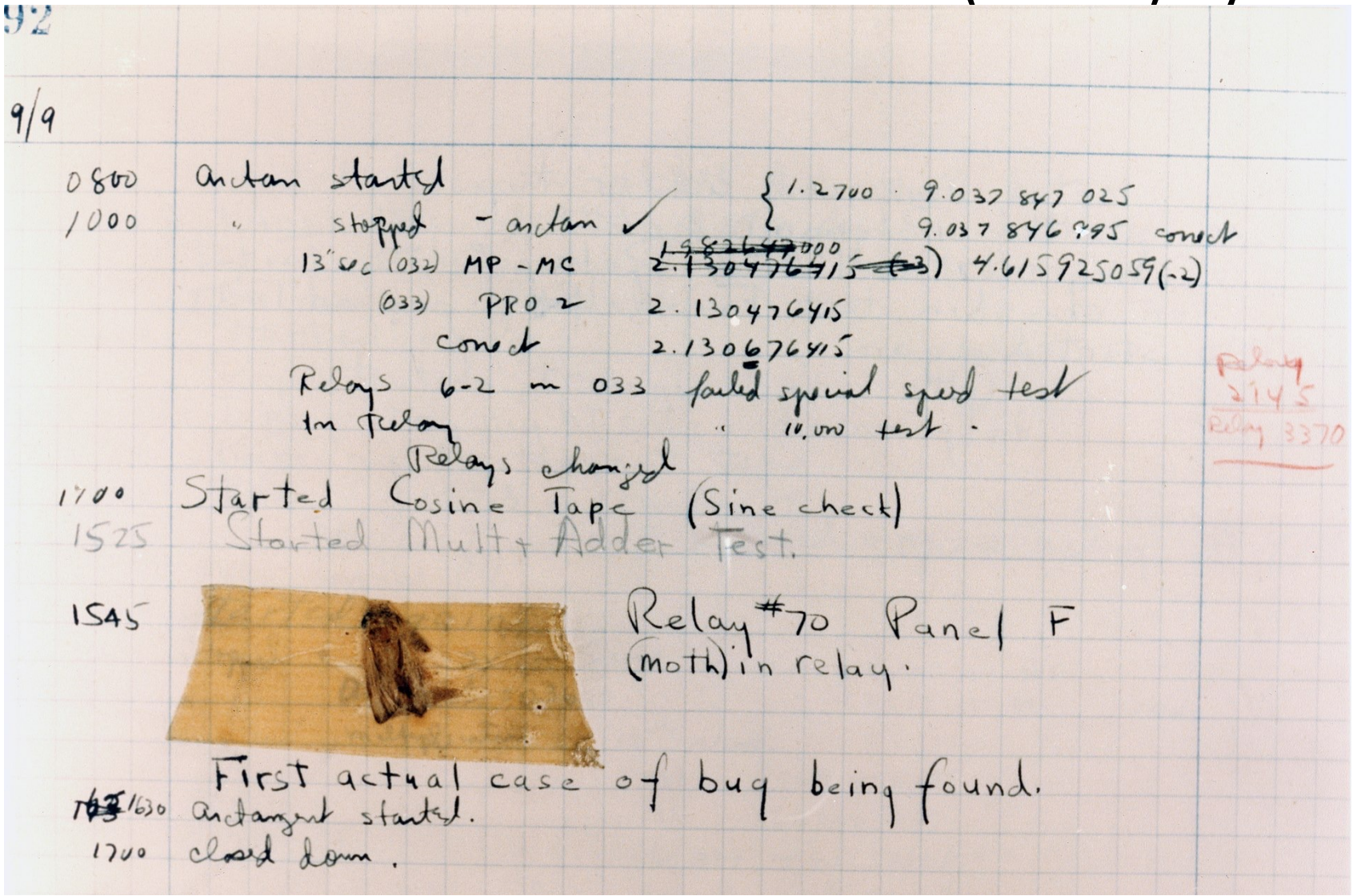
```
prog1.cpp:9:20: error: ‘a’ was not declared in this scope
    9 |         cout << p << a << endl;
      |                    ^
      |
```

Programming Mistakes

- Software "bug"
  - "An error, flaw or fault in the design, development, or operation of computer software that causes it to produce an incorrect or unexpected result, or to behave in unintended ways."
    - en.wikipedia.org/wiki/Software_bug
- "Debugging" = process of finding and correcting bugs
  - Compile-time errors; s++ will usually give line number where it thinks there is an error
  - Run-time errors
- History
  - Middle-English word bugge is the basis for the terms "bugbear" and "bugaboo" as terms used for a monster
  - Term 'bug' used by Thomas Edison (1878), Isaac Asimov (1944)
  - Term 'debug' reported in Oxford Dictionary in context of aircraft engines

Bug

- A page from Harvard Mark II electromechanical computer's log, featuring a dead moth that was removed from the device (1940s) by Admiral Grace Hopper



Bug

- Admiral Grace Hopper
  - American computer scientist, mathematician, US Navy rear admiral
  - Prior to joining the Navy, Hopper earned PhD in mathematics from Yale University
  - Computer programming pioneer
  - Presidential Medal of Freedom

Program Design

- Writing a program to compute: $e = \lim_{n \to \infty} \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$

- Program specification

Input to e computation program: Integer n, where $n \geq 0$.
Output from e computation program: $1/0! + 1/1! + \ldots + 1/n!$

  - Evaluate ‘e’ up to the first (n+1) terms in its expansion
- Observation and planning
  - Sum
    - Need a variable to store sum
    - Summation can be written in a loop; i-th iteration computes i-th term
  - Terms
    - Each term involves a factorial, e.g., i!
    - Need a variable to store each term
    - Computing t! reuses work done to compute (i-1)!, i.e., i! = (i-1)! * i

Program Design

- Plan program structure

```
main_program{
  int n; cin >> n;
  int i = ...;                          // we fill in the blanks later.
  double result = ..., term = ...;

  repeat(n){
    // Need code to calculate 1/0!+1/1!+...+1/t! in the tth iteration.
    // At the beginning of the tth iteration
    // i = t, term = 1/(t-1)!, result = 1/0!+1/1!+...+1/(t-1)!
  }
  cout << result << endl;
}
```

Program Design

- Write the program

```
main_program{
  int n; cin >> n;      // the last term to be added is 1/n!

  int i=1;              // counts iterations of the loop
  double term = 1.0;    // for holding terms of the series
  double result = 1.0; // Will contain the final answer

  repeat(n){ // Plan: When entering for the tth time, t = 1,2,..,n
             // i = t, term = 1/(t-1)!, result = 1/0!+...+1/(t-1)!
    result = result + term/i;
    term = term/i;
    i = i + 1;
  }
  cout << result << endl;
}
```

Program Design

- Test the program
  - Does it compile ? Are there compilation errors ?
    - Syntax errors
  - Does it run without crashing ? Are there run-time errors ?
    - Usually semantic errors
  - Does it run and give the correct output ?
    - Test for various values of n
    - n=0 doesn’t run the loop at all
    - n=1 runs the loop once
    - n=2 runs the loop twice (is term in second run reusing term from first run correctly ?)
    - …
  - Use cout statements profusely during testing; even put them inside loop
  - Comparing against “ground truth”
    - See if we get close to actual value of e for large n

Practice Examples for Lab: Set 3

- 1 done

Write a program to approximately compute e^x by adding first 15 terms of the series

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

done

- 2 Write a program that computes the value of an nth degree polynomial $A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$. Assume that you are given n then the value x, and then the coefficients $a_0, a_1, \dots, a_n$.
- 3 Evaluate the polynomial, but this time assume that you are given the coefficients in the order $a_n, a_{n-1}, \dots, a_0$. done
- 4 Write a program to compute the value of

done

$$D(r) = \sum_{k=0}^{r} (-1)^k \frac{r!}{k!}$$

Practice Examples for Lab: Set 3

done

- 5) Which of the following programs correctly approximates value of ‘e’?

```
main_program{
  int n, fac=1, i=2;
  double e=1.0;
  cin >> n;

  repeat(n){
    e = e + 1.0/fac;
    fac = fac * i;
    i = i + 1;
  }
  cout << e << endl;
}
```

```
main_program{
  int n, fac=1, i=1;
  double e=1.0;
  cin >> n;

  repeat(n){
    e = e + 1.0/fac;
    fac = fac * i;
    i = i + 1;
  }
  cout << e <<
}
```

done

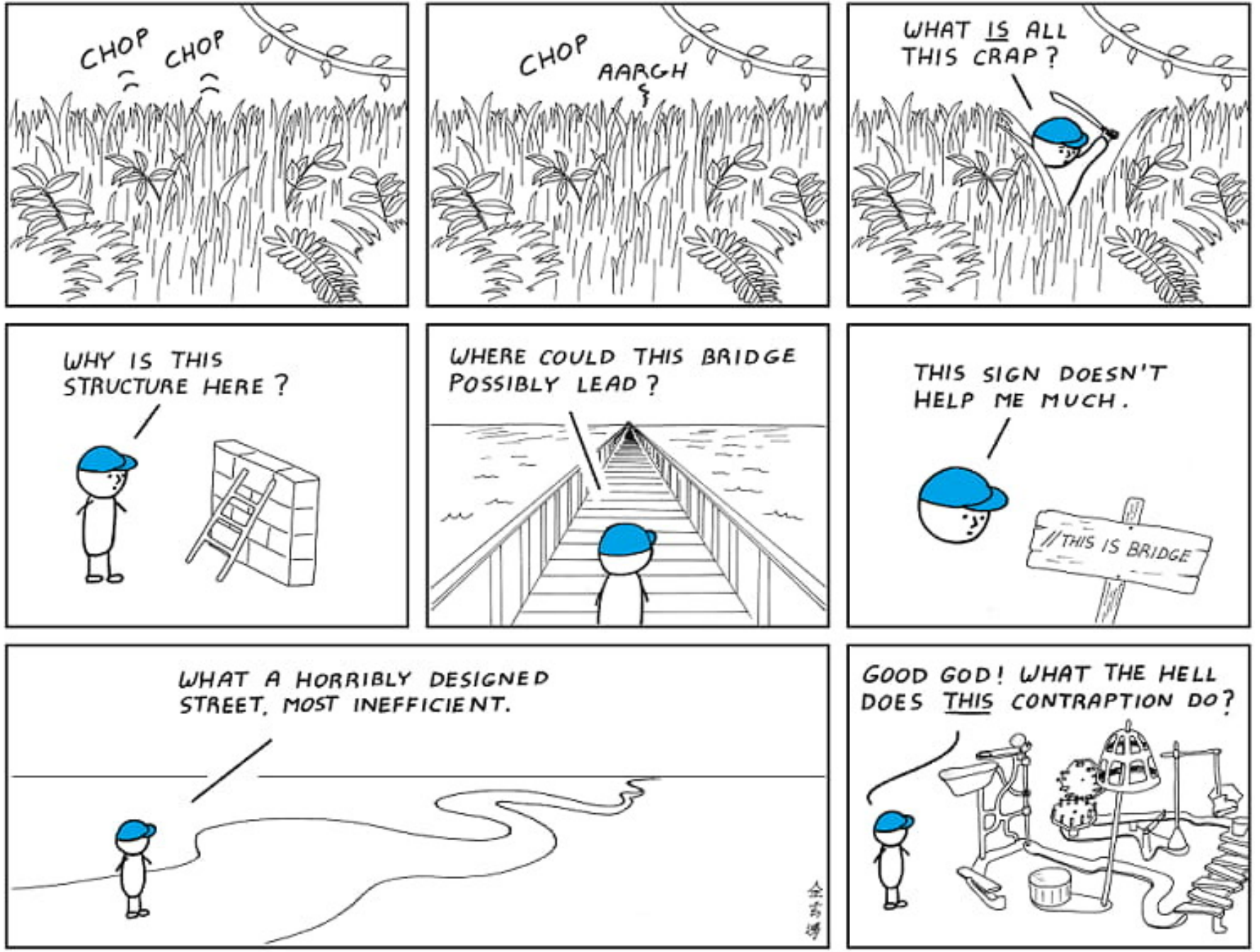
- 6) Write a program to approximately compute sin(x) using its series:

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

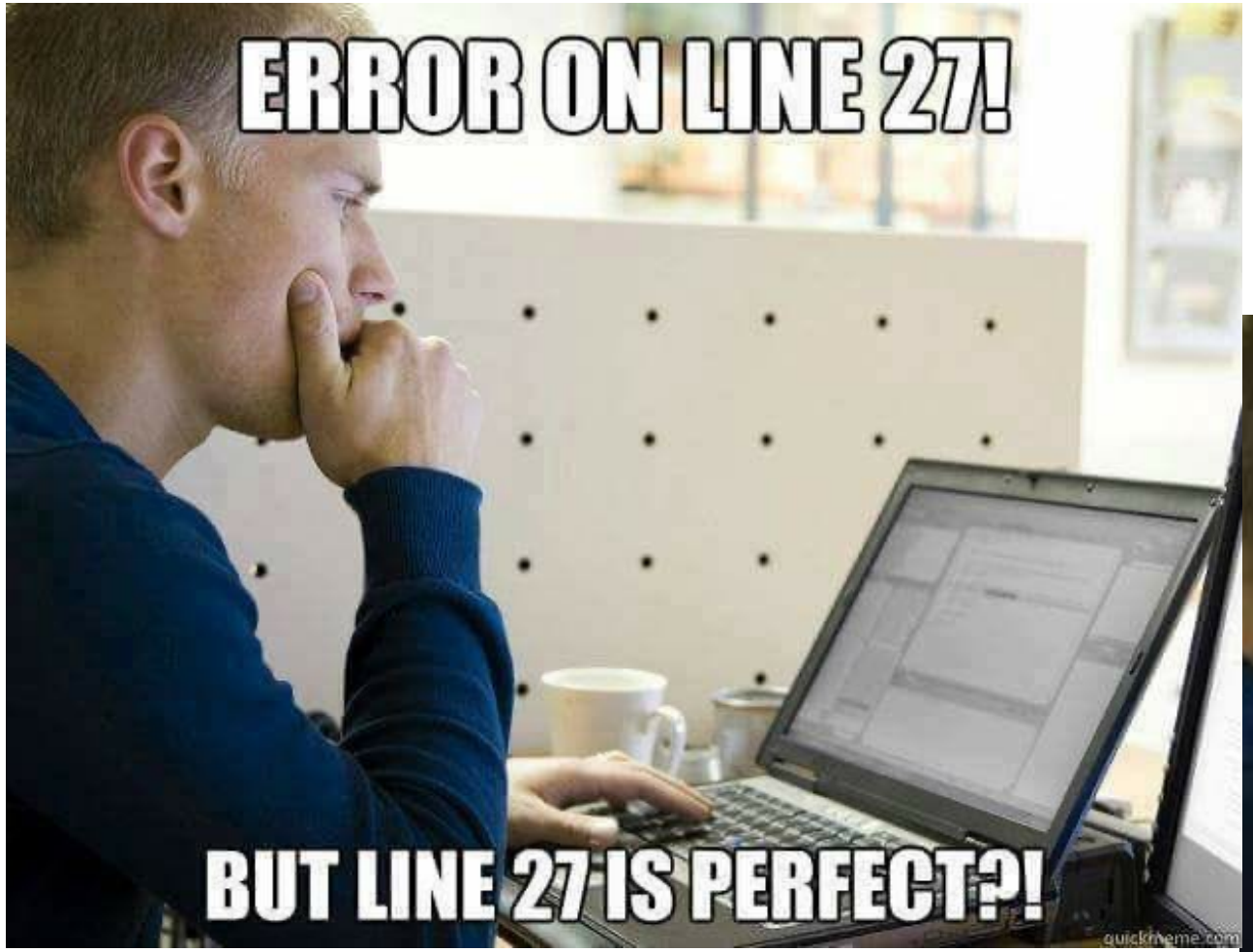
valid for all $x \in \mathbb{R}$.

Program Design



I hate reading other people's code.

Debugging

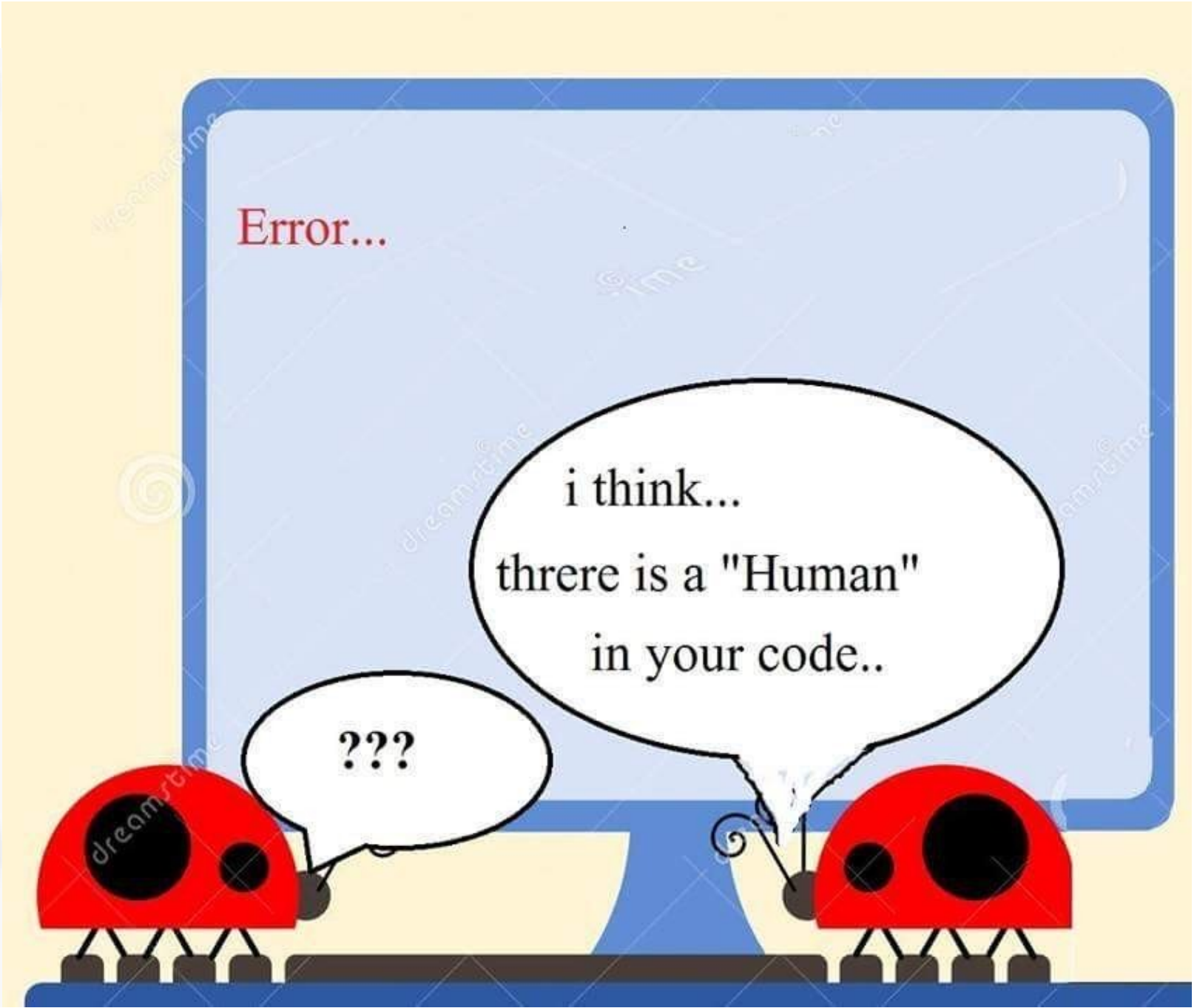

ERROR ON LINE 27!
BUT LINE 27 IS PERFECT?!


Hit Compile. 1 Error. Fix it.
Hit compile again. 927234 errors.

Debugging

6 STAGES OF DEBUGGING

1. That can't happen.
2. That doesn't happen on my machine.
3. That shouldn't happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?

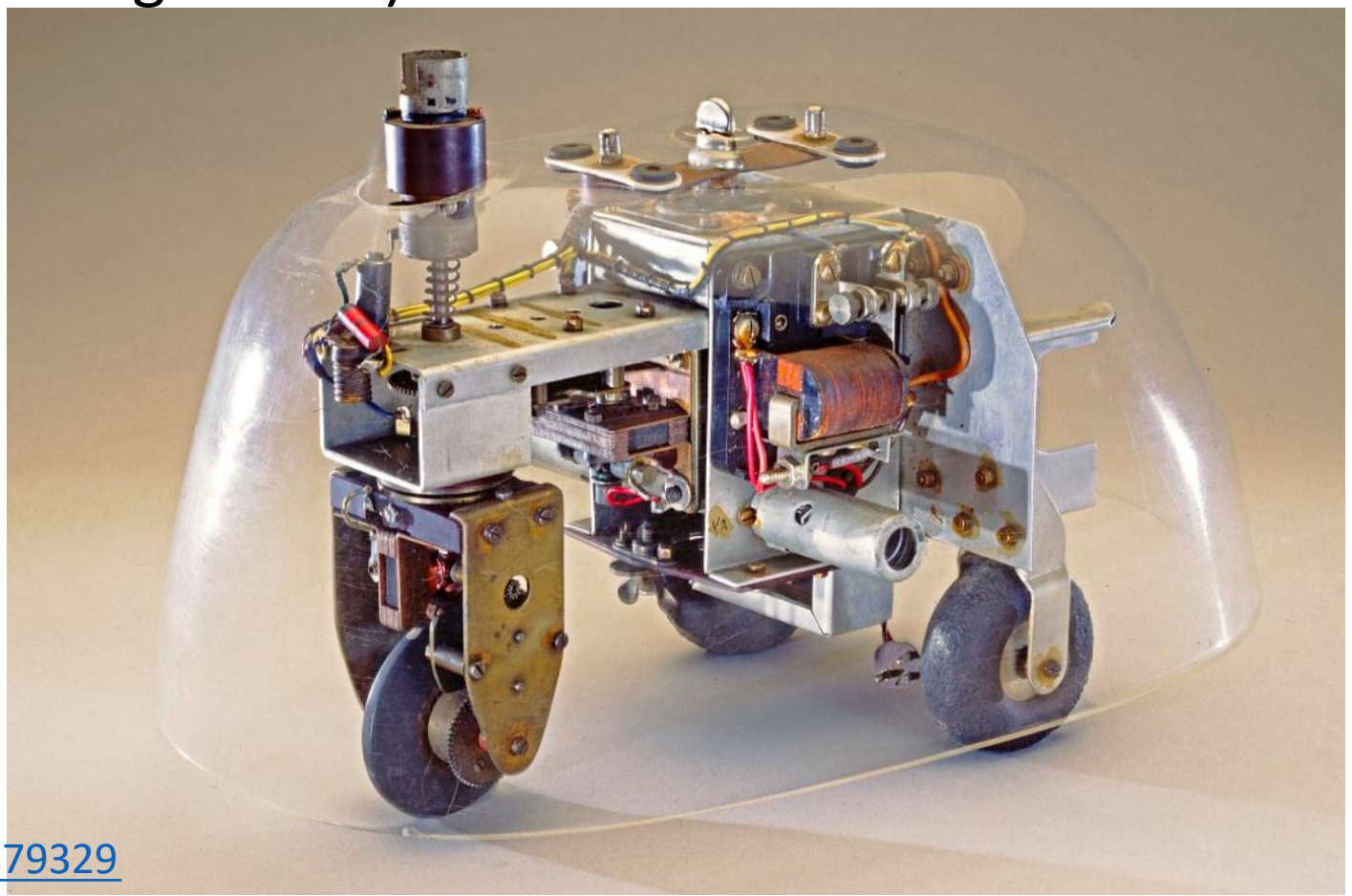


SimpleCPP Graphics

- Let the art begin
- initCanvas()
  - Opens a window, without a turtle
    - Can create a turtle later, if we want
  - Getting the size of canvas: canvas_width(), canvas_height()
  - initCanvas (windowNameString, width, height)
- Canvas coordinates
  - Origin at top left
  - X coordinates increase towards right
  - Y coordinates increase towards bottom
- closeCanvas()
  - removes the window

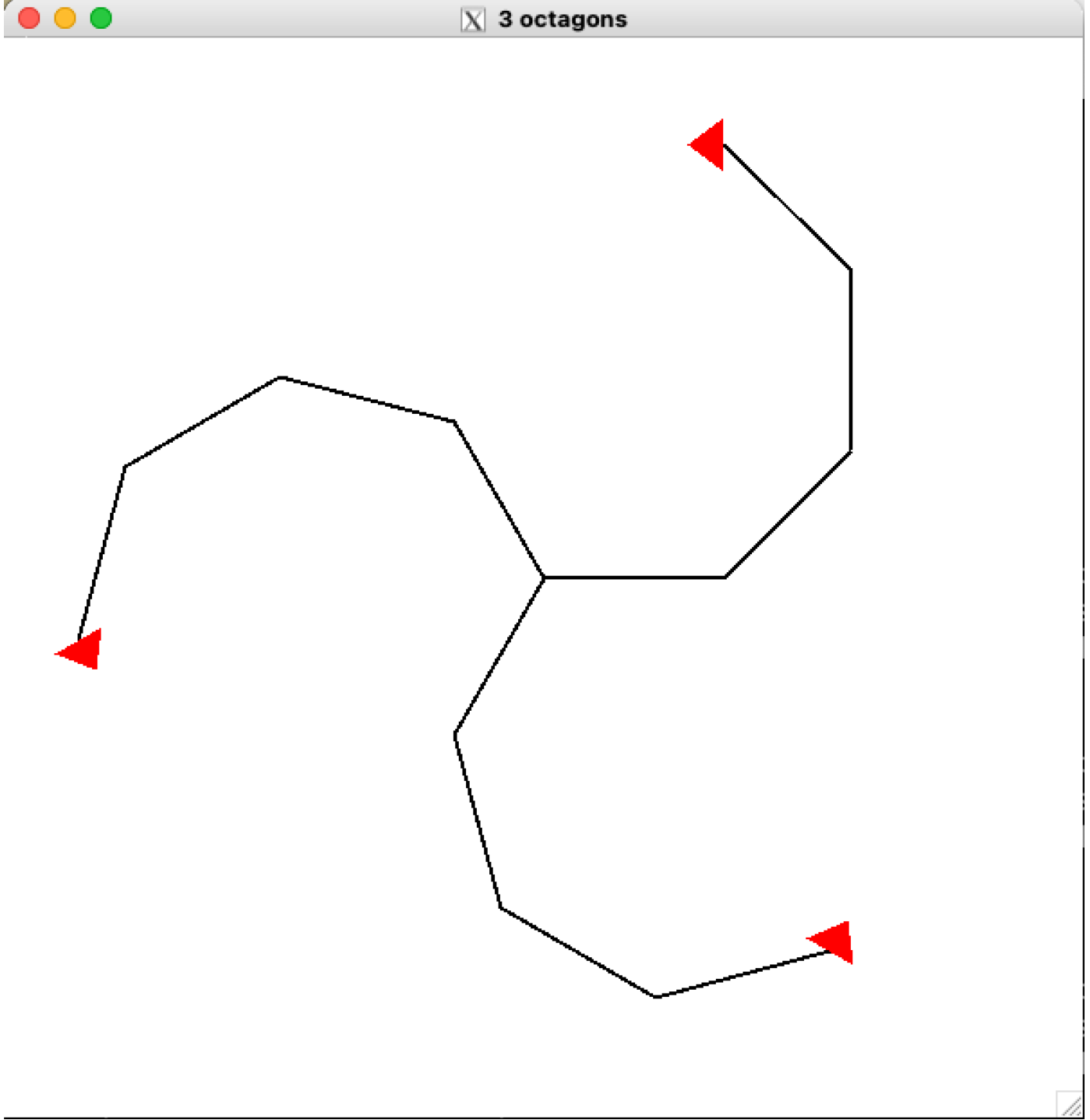
SimpleCPP Graphics

- Multiple turtles !
  - Turtle turtle1, turtle2, turtle3;
  - To command a specific turtle (say, turtle1) use: turtle1.commandName (commandArguments)
    - e.g., turtle1.forward (100);
    - e.g., turtle1.right (90);



americanhistory.si.edu/collections/search/object/nmah_879329

```cpp
#include <simplecpp>

main_program
{
  initCanvas ("3 octagons", 600, 600);

  // create 3 turtles to draw 3 octagons
  Turtle t1, t2, t3;
  // orient each turtle/octagon differently
  t1.left (0);
  t2.left (120);
  t3.left (240);

  repeat (8) // draw octagons
    {
      // draw a side of each of the 3 octagons
      t1.forward (100);
      t2.forward (100);
      t3.forward (100);

      // turn
      t1.left (360 / 8);
      t2.left (360 / 8);
      t3.left (360 / 8);

      // wait for click
      getClick();
    }
}
```



```cpp
#include <simplecpp>

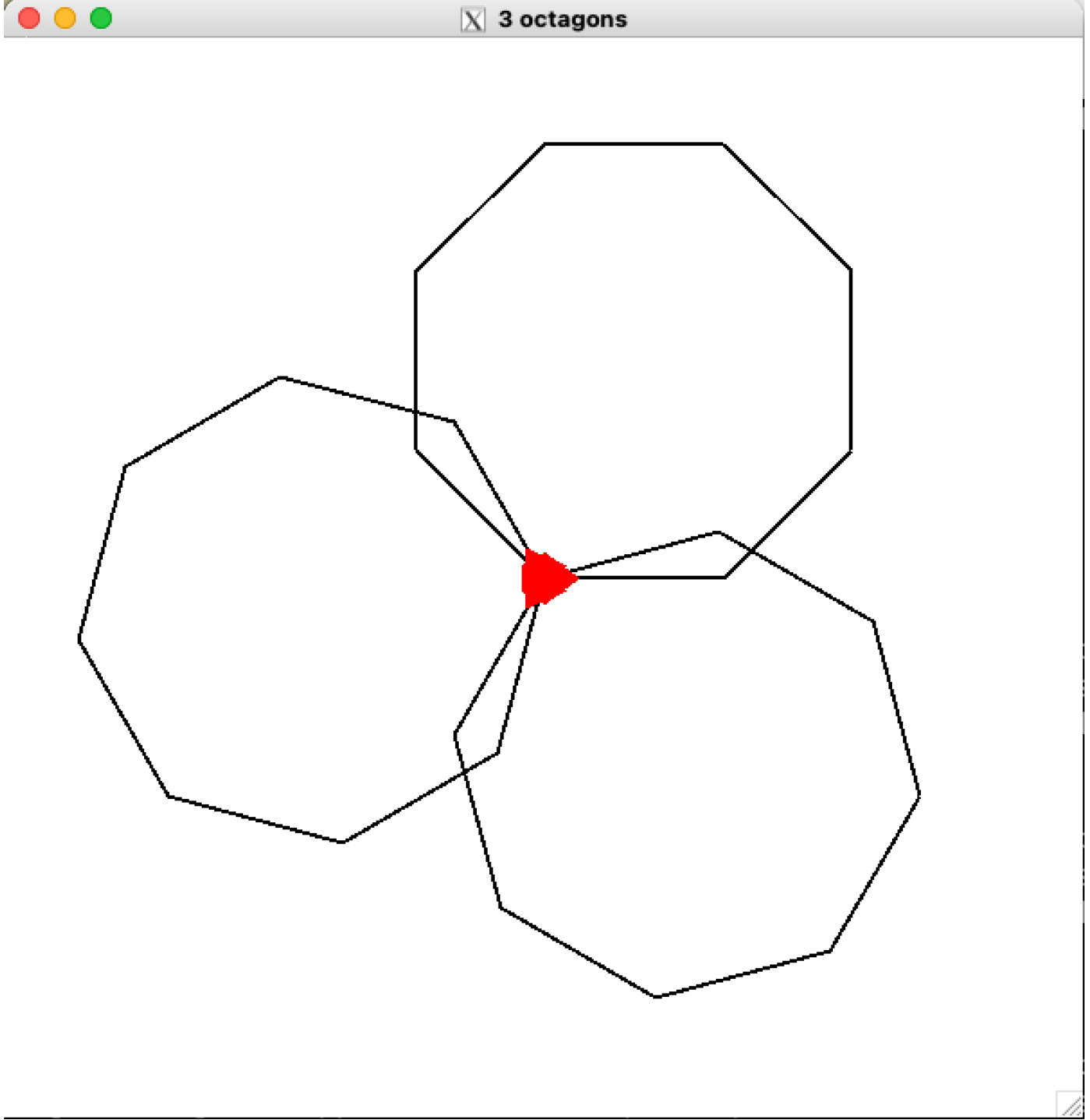
main_program
{
  initCanvas ("3 octagons", 600, 600);

  // create 3 turtles to draw 3 octagons
  Turtle t1, t2, t3;
  // orient each turtle/octagon differently
  t1.left (0);
  t2.left (120);
  t3.left (240);

  repeat (8) // draw octagons
    {
      // draw a side of each of the 3 octagons
      t1.forward (100);
      t2.forward (100);
      t3.forward (100);

      // turn
      t1.left (360 / 8);
      t2.left (360 / 8);
      t3.left (360 / 8);

      // wait for click
      getClick();
    }
}
```



SimpleCPP Graphics

- Other shapes
  - Circle circleName (centerCoordinateX, centerCoordinateY, radius);
    - e.g., Circle c1 (10, 10, 100);
    - All arguments of type double
  - Rectangle rectangleName (centerX, centerY, width, height);
    - e.g., Rectangle r1 (0, 0, 100, 200);
  - Line lineName (lineStartX, lineStartY, lineEndX, lineEndY);
    - e.g., Line l1 (0, 0, 100, 200);
  - Text textName (centerX, centerY, messageString);
    - e.g., Text t1 (10, 20, “Hello C++”)
    - textWidth (“Hello C++”), textHeight (“Hello C++”) gets width and height of text
    - ```
      Text t(100,100,"C++ g++");
      Rectangle R(100,100,textWidth("C++ g++"),textHeight());
      ```
      ```

SimpleCPP Graphics

- Commands allowed on a shape (say, s)

```
s.moveTo(x,y);
s.move(dx,dy);
```

where the former moves the shape to coordinates `(x,y)` on the screen, and the latter displaces the shapes by `(dx,dy)` from its current position.

```
s.scale(relfactor);
s.setScale(factor);
```

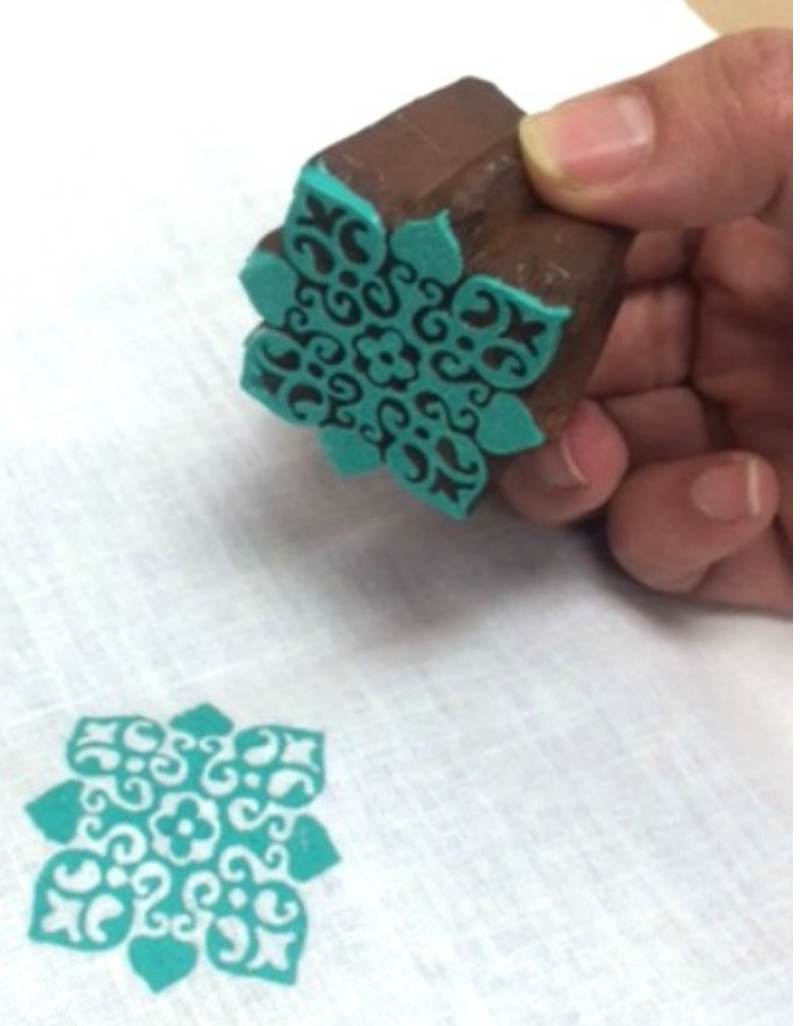
Here `relfactor, factor` are expected to be `double`. The first version multiplies the current scale factor by the specified `relfactor`, the second version sets the scale factor to `factor`.

SimpleCPP Graphics

- Commands allowed on a shape (say, s)
  - s.setColor (color)
    - Color argument can be COLOR(“red”) or COLOR(redValue, greenValue, blueValue)
  - s.setFillColor (COLOR(“red”)), s.setFill (booleanVar)
    - Does not apply to line shapes
  - s.hide()
  - s.show()
  - s.rotate(angle)
    - Angle should be in radians, measured clockwise
- Getting information about a shape
  - s.getX(), s.getY()
  - s.getScale() – returns scale factor
  - s.getOrientation() – returns angle

SimpleCPP Graphics

- Imprinting shapes on to canvas
  - s.imprint()
- Drawling lines without creating a line shape
  - imprintLine (lineStartX, lineStartY, lineEndX, lineEndY, color);
  - imprintLine (lineStartX, lineStartY, lineEndX, lineEndY);
    - Default color is black
- Resetting a shape
  - s.reset(arguments same as those used during creation)



SimpleCPP Graphics

- Clicks from the canvas
  - getClick() waits for the user to click at some coordinate on the canvas;
  - When user clicks at location (x,y), then getClick() returns integer v=65536*x+y
  - (x,y) can be recovered from v as: $x = \lfloor v/65536 \rfloor, \qquad y = v \bmod 65536$
  - Note:
    v is a 32-bit integer.
    2^{16} = 65536.
    Assuming coordinates x,y << 65536; so 16-bits is enough to store them.
    So, 65536*x is a number that has all zeros in least-significant 16 bits.
    So, x gets stored in most-significant 16 bits
    and y gets stored in least-significant 16 bits.

SimpleCPP Graphic

- Shooting a projectile on canvas using shapes and getClick()

```
#include <simplecpp>

main_program{
  initCanvas("Projectile motion", 500,500);

  int start = getClick();

  Circle projectile(start /65536, start % 65536, 5);
  projectile.penDown();

  double vx=1,vy=-5, gravity=0.1;

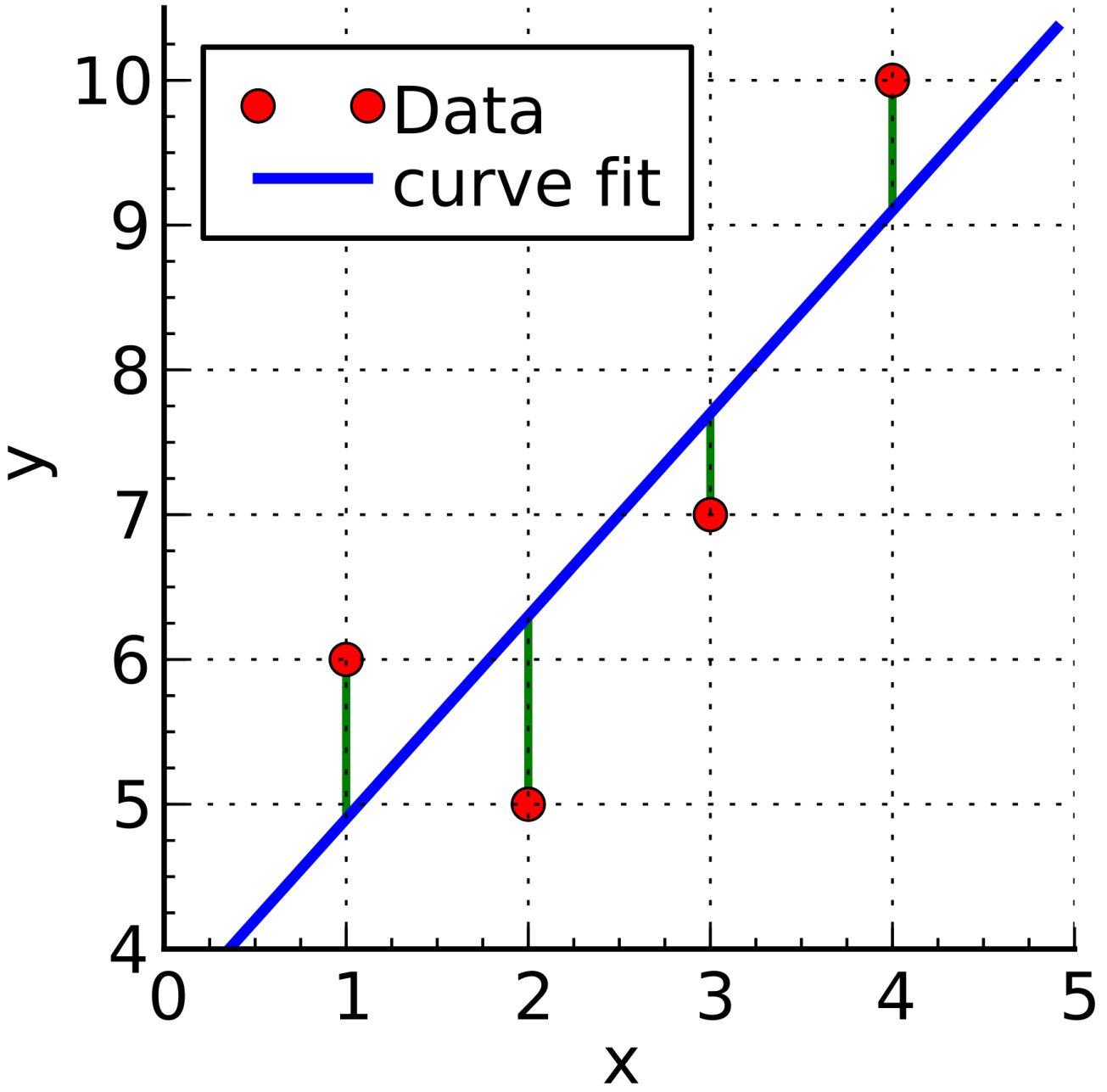
  repeat(100){
    projectile.move(vx,vy);
    vy += gravity;
    wait(0.1);
  }
  wait(10);
}
```

Line Fitting (Linear Regression)

- Data: $(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)$
- Find line: $y = mx + c$
- Define quality of fit as negative sum of squared vertical distances
- Find line maximizing quality of fit

$$\min \sum_{i=1}^{n} (y_i - mx_i - c)^2$$

  - Minimize over variables m and c
- Differentiate w.r.t. m and c, assign derivatives to zero, solve 2 equations in 2 unknowns



Line Fitting (Linear Regression)

- Differentiate w.r.t. m and c, assign derivatives to zero, solve 2 equations in 2 unknowns

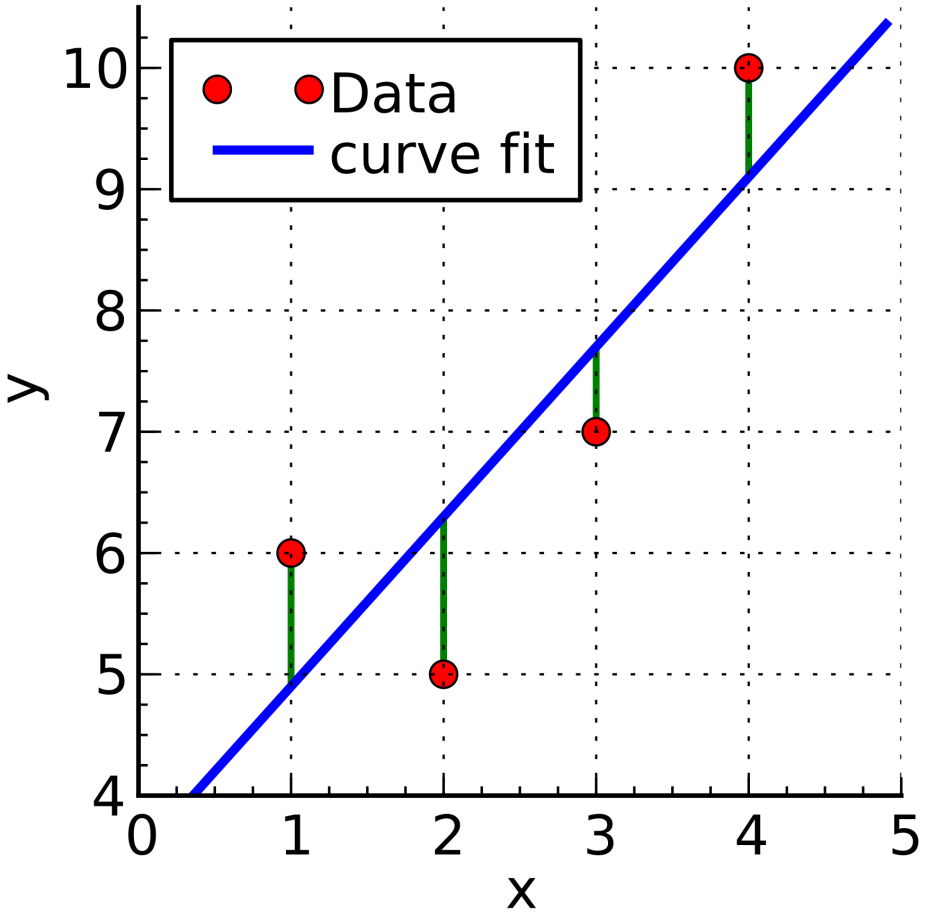
$$0 = \frac{\partial}{\partial m}\sum_{i=1}^{n}(y_i - mx_i - c)^2 = -2\sum_{i=1}^{n} x_i(y_i - mx_i - c)$$

$$m\sum_i x_i^2 + c\sum_i x_i = \sum_i x_i y_i$$

$$0 = \frac{\partial}{\partial c}\sum_{i=1}^{n}(y_i - mx_i - c)^2 = -2\sum_{i=1}^{n}(y_i - mx_i - c)$$

$$m\sum_i x_i + nc = \sum_i y_i$$

$p = \sum_i x_i^2,\ q = \sum_i x_i,\ r = \sum_i x_i y_i,$ and $s = \sum_i y_i$



$$m = \frac{nr - qs}{np - q^2} \qquad c = \frac{ps - qr}{np - q^2}$$

```
#include <simplecpp>

main_program
{
  cout << "Number of data points: ";
  unsigned int n; cin >> n;

  initCanvas ("Line Fitting", 600, 600);

  Circle shapeCircle(0,0,0); // to show the data on the canvas

  double p = 0, q = 0, r = 0, s = 0; // internal variables

  repeat (n) // get data, computer internal variables
  {
    // get the datum coordinates
    const int codedLocation = getClick();
    const double x = codedLocation / 65536;
    const double y = codedLocation % 65536;

    // plot the datum
    shapeCircle.reset (x, y, 5);
    shapeCircle.imprint ();

    p += x*x;
    q += x;
    r += x*y;
    s += y;
  }

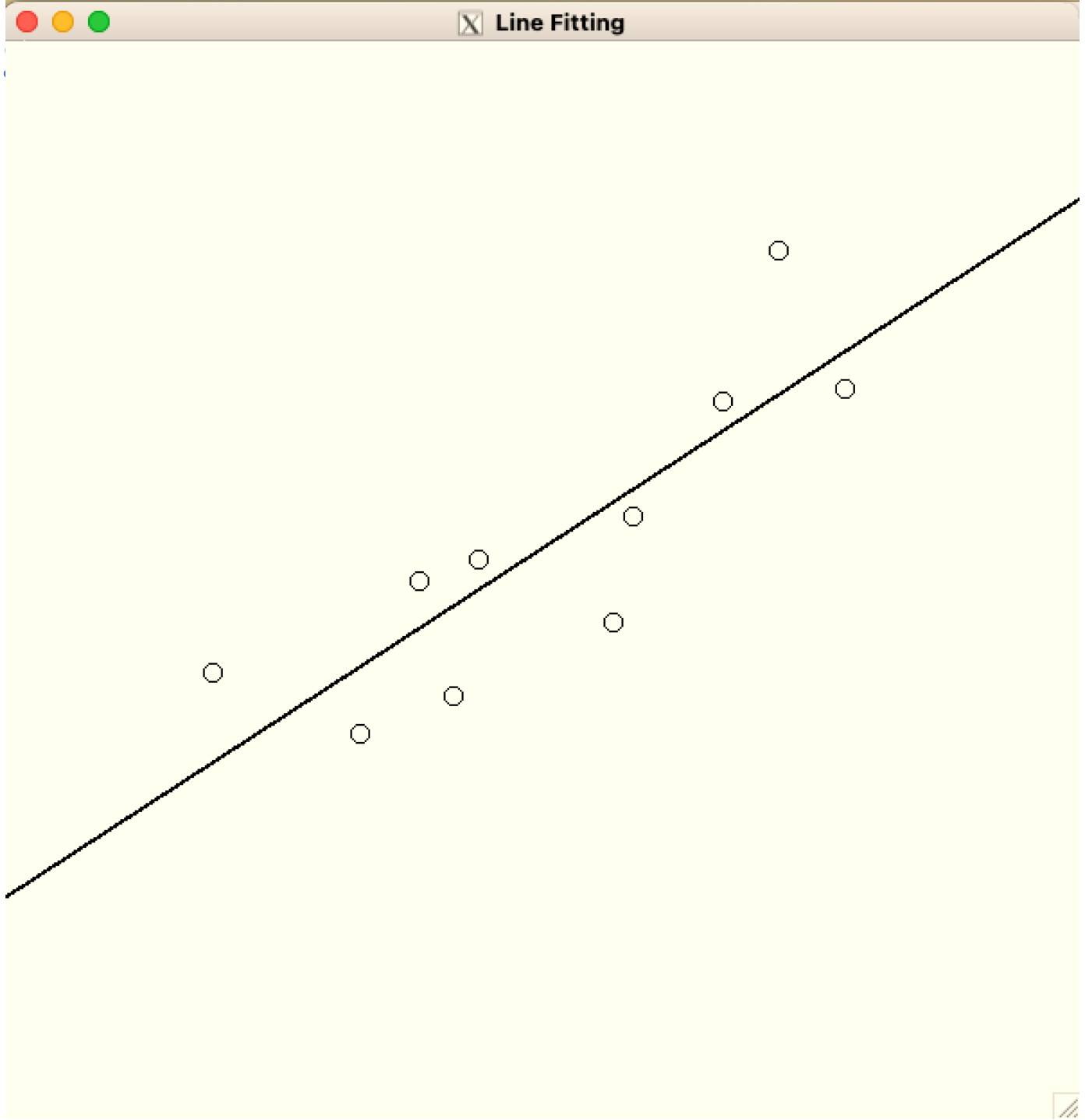
  // find the best-fit line
  const double lineSlope     = (n*r - q*s) / (n*p - q*q);
  const double lineIntercept = (p*s - q*r) / (n*p - q*q);

  // plot the best-fit line
  Line fittedLine (0, lineIntercept, 600, 600 * lineSlope + lineIntercept);

  getClick();
}
```

Line Fitting (Linear Regres

- Program run with input as 10 datapoints



Pseudocode

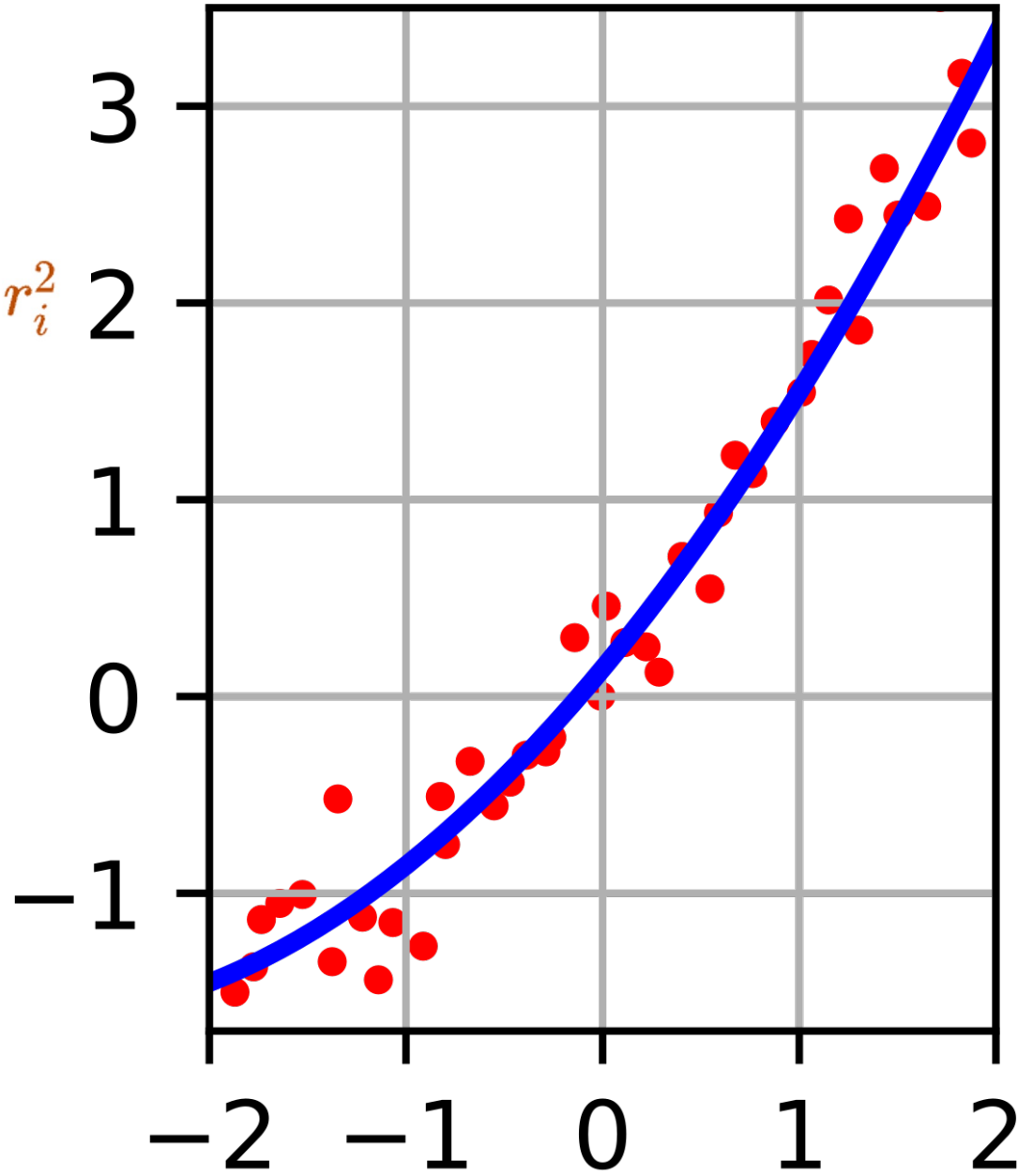
- Definition: Plain-language description of the steps in an algorithm
- Often uses structural conventions of a normal programming language, but is intended for human reading rather than machine reading
  - Uses natural language; uses compact mathematical notation
  - Omits details essential for machine understanding of algorithm, e.g., variable definitions and language-specific code
- Pseudocode is subjective; a standard doesn't exist
- Example
  - Set sum = 0
    Repeat 5 times
      Get number
      sum = sum + number^2
    Print sum

Regression

- Statistical analysis
- Method of Least Squares
  - Minimize sum of squared errors $S = \sum_{i=1}^{n} r_i^2$
    - Error between value predicted by model and datum

$$r_i = y_i - f(x_i, \boldsymbol{\beta})$$

  - Described by Legendre in 1805
  - Described by Gauss in 1809 (or 1795)
    - Used in astronomy



Practice Examples for Lab: Set 4

- 1

Plot the graph of $y = \sin(x)$ for x ranging in the interval 0 to 4π. Draw the axes and mark the axes at appropriate points, e.g. multiples of $\pi/2$ for the x axis, and multiples of 0.25 for the y axis.

- 2

Draw an 8×8 chessboard having red and blue squares. Hint: Use the `imprint` command. Use the repeat statement properly so that your program is compact.

- 3

Modify the projectile motion program so that the velocity is given by a second click. The projectile should start from the first click, and its initial velocity should be in the direction of the second click (relative to the first). Also the velocity should be taken to be proportional to the distance between the two clicks.

- 4

Write a program that accepts 3 points on the canvas (given by clicking) and then draws a circle through those 3 points.

Practice Examples for Lab: Set 4

- 5

In this problem you are to determine how light reflects off a perfectly reflecting spherical surface. Suppose the sphere has radius r and is centered at some point (x, y). Suppose there is a light source at a point (x', y). Rays will emerge from the source and bounce off the sphere. As you may know, the reflected ray will make an angle to the radius at the point of contact equal to that made by the incident ray. Write a program which traces many such rays. It should take r, x, y, x' as input. Of course, in the plane the sphere will appear as a circle.

- 6

This is an extension to the previous problem. Extend the reflected rays backward till they meet the line joining the circle center and light source. The points where the rays meet this line can be said to be the image of the light source in the mirror; as you will see this will not be a single point, but the image will be diffused. This is the so called spherical aberration in a circular mirror.

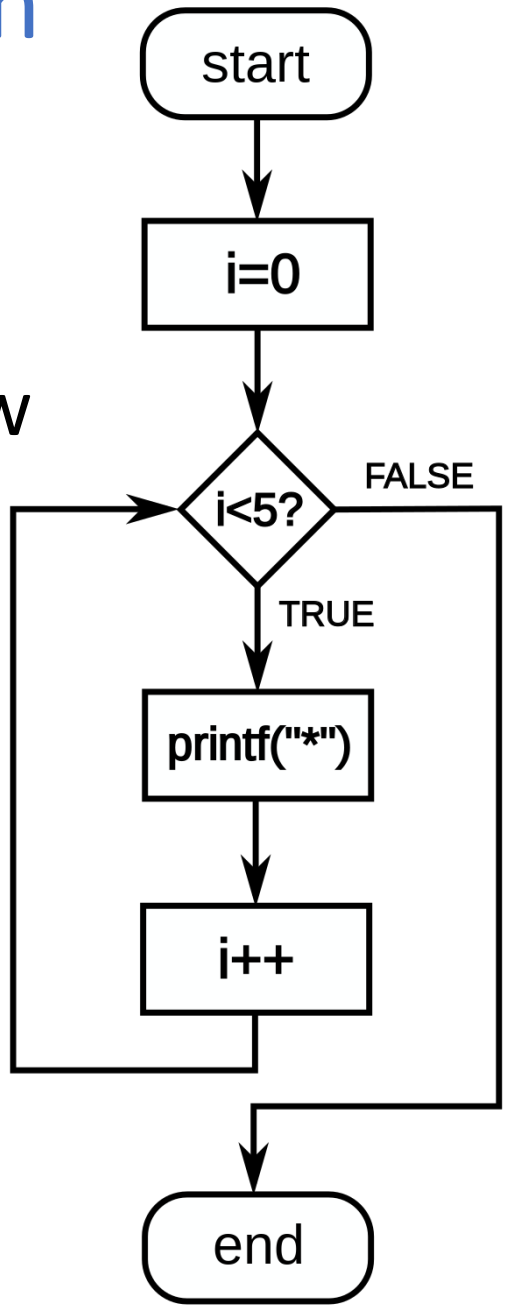
CS 101
Computer Programming and Utilization

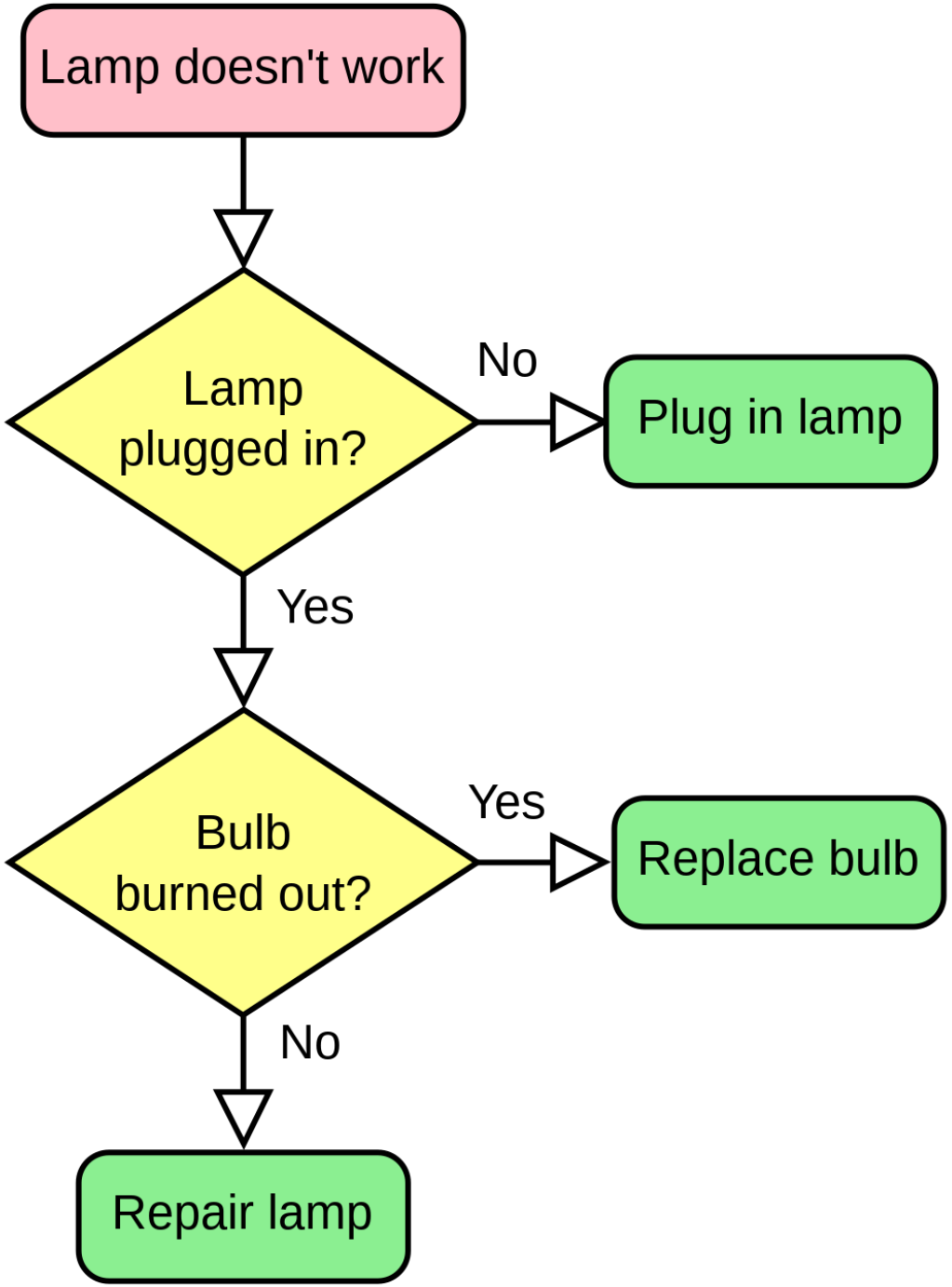
Conditions, Loops

Suyash P. Awate

Conditional Execution

- Flowcharts
  are often based on
  conditions that
  direct/change control flow
  leading to
  conditional execution
  of instructions/code

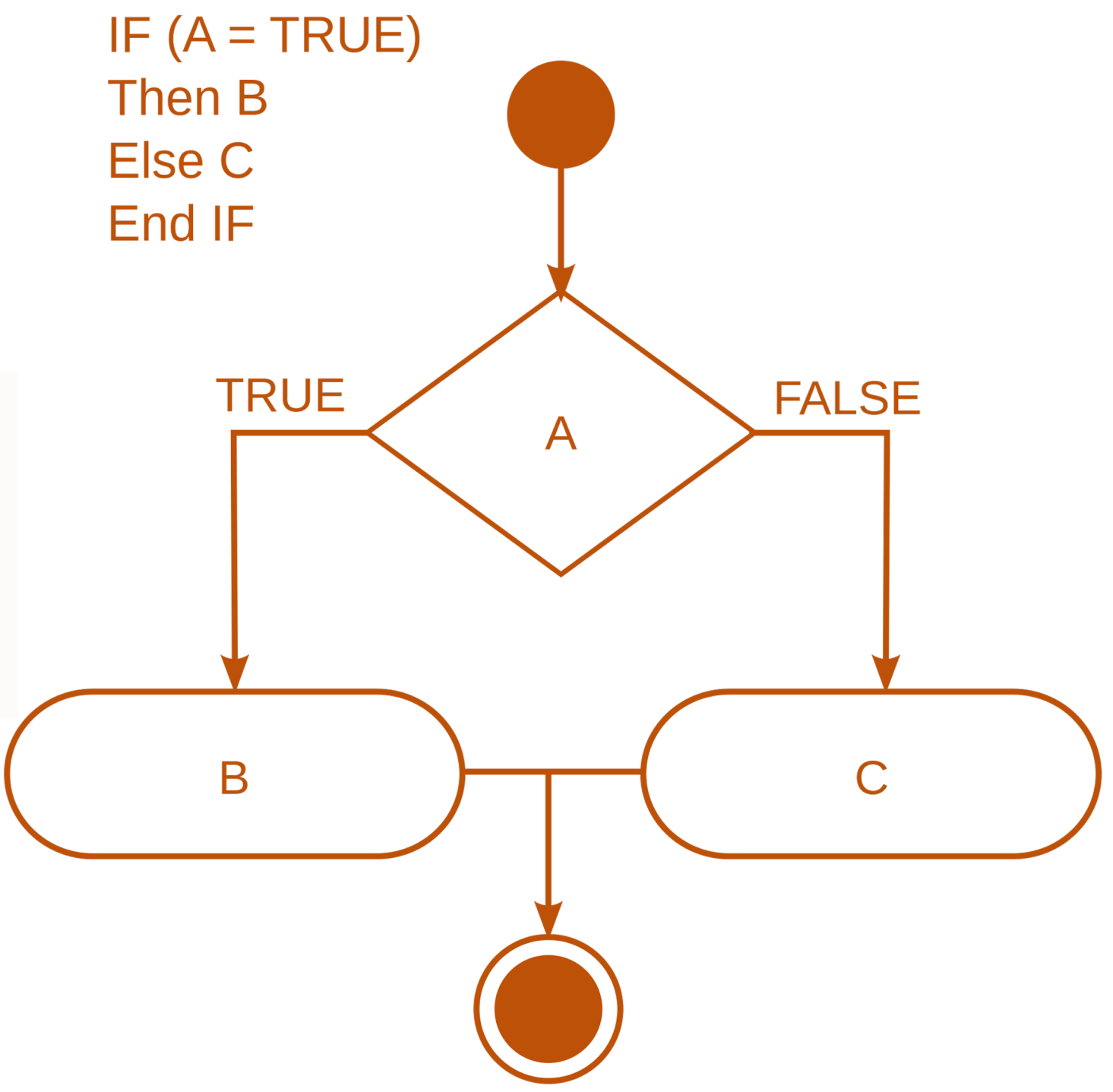




Conditional Execution

- Pseudocode describing an "if statement" in typical programming languages

```
If (boolean condition) Then
   (consequent)
Else
   (alternative)
End If
```



Conditional Execution

- In Simplecpp
  - if (condition) consequent
  - if (condition) consequent else alternative
- Condition: a simple form
  - Expression1 relationalOperator Expression2
- Relational operators: <, >, <=, >=, ==, !=
- Condition: examples
  - “A == 5”
  - “A < 5”
  - “A != 5”
  - If value of A is 5, then first condition **evaluates** to boolean “true”, and other conditions evaluate to boolean “false”

Conditional Execution

- Condition: more complex forms
  - Conjunction (“and”) of multiple conditions
    - e.g., condition1 && condition2
    - && = logical AND operator; applies to Boolean variables
  - Disjunction (“or”) of multiple conditions
    - e.g., condition1 || condition2 || condition3
    - || = logical OR operator; applies to Boolean variables
  - Negation
    - !condition
    - ! = negation operator; applies to Boolean variables

Conditional Execution

- Consequent and alternative can be single statements or blocks
- Example
  - if (a == 5)
    {
      statement 1;
      statement 2;
    }

Conditional Exec

- Example1
  - Income tax calculation

```
main_program{
  float income; // in rupees.
  float tax;    // in rupees.

  cout << "What is your income in rupees? ";
  cin >> income;

  if(income <= 180000) tax = 0;                          //
  if((income >  180000) && (income <= 500000))           //
    tax = (income - 180000)* 0.1;
  if((income >  500000) && (income <= 800000))           //
    tax = 32000+(income - 500000)* 0.2;
  if(income > 800000)                                    //
    tax = 92000+(income - 800000)* 0.3;

  cout << "Tax is: " << tax << endl;
}
```

Conditional Execution

- Simplecpp if statement: complex form

```
if (condition1) consequent1
else if (condition2) consequent2
else if (condition3) consequent3
...
else if (conditionn) consequentn
else alternate
```

Conditional Execution

- Example2
  - Income tax calculation
- Which one executes faster: example1 or example2 ?

```
main_program{
  float income, tax;

  cout << "What is your income? ";
  cin >> income;

  if(income <= 180000) tax = 0;          //
  else if(income <= 500000)              //
    tax = (income - 180000)* 0.1;
  else if(income <= 800000)              //
    tax = 32000+(income - 500000)* 0.2;
  else
    tax = 92000+(income - 800000)* 0.3;


  cout << "Tax is: " << tax << endl;
}
```

Conditional Execution

- Turtle controller

```
main_program{
  char command;
  turtleSim();

  repeat(100){
    cin >> command;
    if (command == 'f') forward(100);
    else if (command == 'r') right(90);
    else if (command == 'l') left(90);
    else cout << "Not a proper command, " << command << endl;
  }
}
```

Conditional Execution

- Building buttons
- Detecting clicks on a button

```
main_program{
  initCanvas();

  const float bFx=150,bFy=100, bLx=400,bLy=100, bWidth=150,bHeight=50;
  Rectangle buttonF(bFx,bFy,bWidth,bHeight), buttonL(bLx,bLy,bWidth,bHeight);

  Text tF(bFx,bFy,"Forward"), tL(bLx,bLy,"Left Turn");
  Turtle t;

  repeat(100){
    int clickPos = getClick();
    int cx = clickPos/65536;
    int cy = clickPos % 65536;

    if(bFx-bWidth/2<= cx && cx<= bFx+bWidth/2 &&
       bFy-bHeight/2 <= cy && cy <= bFy+bHeight/2) t.forward(100);

    if(bLx-bWidth/2<= cx && cx<= bLx+bWidth/2 &&
       bLy-bHeight/2 <= cy && cy <= bLy+bHeight/2) t.left(10);
  }
}
```

Conditional Execution

- Switch statement (helps avoid too many if-then-else statements)
  - Intended to be easier to read, understand, maintain, debug
  - "expression", constant1, constant2, … must be of type integer (int/char)
  - First evaluate expression
  - Second
    If value = constantX, then control passes to statements group(X)
    - If "break" not present, execution falls through the next group
    - Not having a "break" can make code complex/confusing; try to avoid
  - If value doesn't match any constant, default group executed

```
switch (expression){
  case constant1:
     group(1) of statements usually ending with ‘‘break;’’
  case constant2:
     group(2) of statements usually ending with ‘‘break;’’
  ...
  default:
     default-group of statements
}
```

Conditional Execution

- Switch

```
main_program{
  char command;
  turtleSim();

  repeat(100){
    cin >> command;
    switch(command){
      case 'f': forward(100);
                break;
      case 'r': right(90);
                break;
      case 'l': left(90);
                break;
      default: cout << "Not a proper command, " << command << endl;
    }
  }
}
```

Conditional Execution

- Switch without break statements in some cases

```
main_program{
  int month;
  cin >> month;
  switch(month){
    case 1:  // January
    case 3:  // March
    case 5:  // May
    case 7:  // July
    case 8:  // August
    case 10: // October
    case 12: // December
             cout << ‘‘This month has 31 days.\n’’;
             break;
    case 2:  // February
             cout << ‘‘This month has 28 or 29 days.\n’’;
             break;
    case 4:  // April
    case 6:  // June
    case 9:  // September
    case 11: // November
             cout << ‘‘This month has 30 days.\n’’;
             break;
    default: cout << ‘‘Invalid input.\n’’;
  }
}
```

Conditional Execution

- Conditional expression

```
condition ? consequent-expression : alternate-expression
```

  - First evaluates condition
  - If condition evaluates to true, then consequent-expression is evaluated
    - Value of consequent-expression becomes value of conditional expression
  - If condition evaluates to false, then alternate-expression is evaluated
    - Value of alternate-expression becomes value of conditional expression
  - Example

```
int marks; cin >> marks;
int actualmarks = (marks > 100) ? 100 : marks;
char grade = (marks >= 35) ? 'p' : 'f';
```

Conditional Execution

- Conditional expressions can be nested
  - Difficult to read and interpret; avoid

```
if(income <= 180000) tax = 0;          //
else if(income <= 500000)              //
  tax = (income - 180000)* 0.1;
else if(income <= 800000)              //
  tax = 32000+(income - 500000)* 0.2;
else
  tax = 92000+(income - 800000)* 0.3;
```

```
main_program{
  float income; cin >> income;

  cout << (                                  //  **
          income <= 180000 ? 0 :
          income <= 500000 ? (income - 180000) * 0.1 :
          income <= 800000 ? 32000 + (income - 500000) * 0.2 :
          92000 + (income - 800000) * 0.3
         )                                   //  **
       << endl;
}
```

Conditional Execution

- Logical variables and "data"
  - Data can have Boolean (categorical) values
  - Example
    - Logical variable indicating low/mid/high income

```
float income; cin >> income;
bool lowIncome, midIncome, highIncome;
lowIncome = (income <= 180000);
midIncome = (income > 180000) && (income <= 800000);
highIncome = (income > 800000);
```

    - Recall: conditions evaluate to boolean true or false
    - Suppose income had value 200,000. Then what would be values for bool variables ?
  - Example
    - Logical variable indicating letter case

```
char in_ch;
bool lowerCase;
cin >> in_ch;
lowerCase = (in_ch >= 'a') && (in_ch <= 'z');
```

    - "cout << true;" outputs "1" on the screen; "cout << false;" outputs "0" on the screen.

Conditional Execution

- For any logical value b
  - b && true = b
  - b || false = b
- Distributivity of && (conjunction) over || (disjunction)
  - Similar to those in logic
    - https://en.wikipedia.org/wiki/Distributive_property#Propositional_logic

$$
\begin{array}{lll}
(P \wedge (Q \vee R)) & \Leftrightarrow ((P \wedge Q) \vee (P \wedge R)) & \text{Distribution of conjunction over disjunction} \\
(P \vee (Q \wedge R)) & \Leftrightarrow ((P \vee Q) \wedge (P \vee R)) & \text{Distribution of disjunction over conjunction} \\
(P \wedge (Q \wedge R)) & \Leftrightarrow ((P \wedge Q) \wedge (P \wedge R)) & \text{Distribution of conjunction over conjunction} \\
(P \vee (Q \vee R)) & \Leftrightarrow ((P \vee Q) \vee (P \vee R)) & \text{Distribution of disjunction over disjunction}
\end{array}
$$

- De Morgan's laws
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

Conditional Execution

- Augustus De Morgan
  - British mathematician, logician
  - Born in Madurai
  - Introduced mathematical induction
  - In 1823, at age 16, entered Cambridge Univ.
  - At age 22, professor of math at London Univ.
- London Univ. established on ideas on religious neutrality
  - “Oxford and Cambridge were so guarded by theological tests that no Jew or Dissenter outside the Church of England could enter as a student, still less be appointed to any office”
    - en.wikipedia.org/wiki/Universities_Tests_Act_1871
    - ox.ac.uk/about/oxford-people/women-at-oxford/centenary-womens-timeline

Conditional Execution

- Is a number prime
- How can you improve this code ?
  - Repeating (x-2) times is redundant
  - Need a way to stop repeat loop when factor is found

```
main_program{  //Decide if x is prime.
   int x; cin >> x;

   int i=2;
   bool found = false;
   repeat(x-2){
      found = found || (x % i) == 0;
      i = i+1;
   }

   if (found) cout << x << " is composite." << endl;
   else cout << x << " is prime." << endl;
}
```

Conditional Execution

- Be careful with “=” and “==”
  - Consider
    - if (a == 5) cout << “found”;
    - if (a = 5) cout << “found”;
  - What will be the result of executing both statements ?
  - Expression “a=5” evaluates to 5, which will then be converted to Boolean
    - Rule for conversion:
      any non-zero integer value converts to Boolean true value;
      a zero integer value converts to Boolean false value

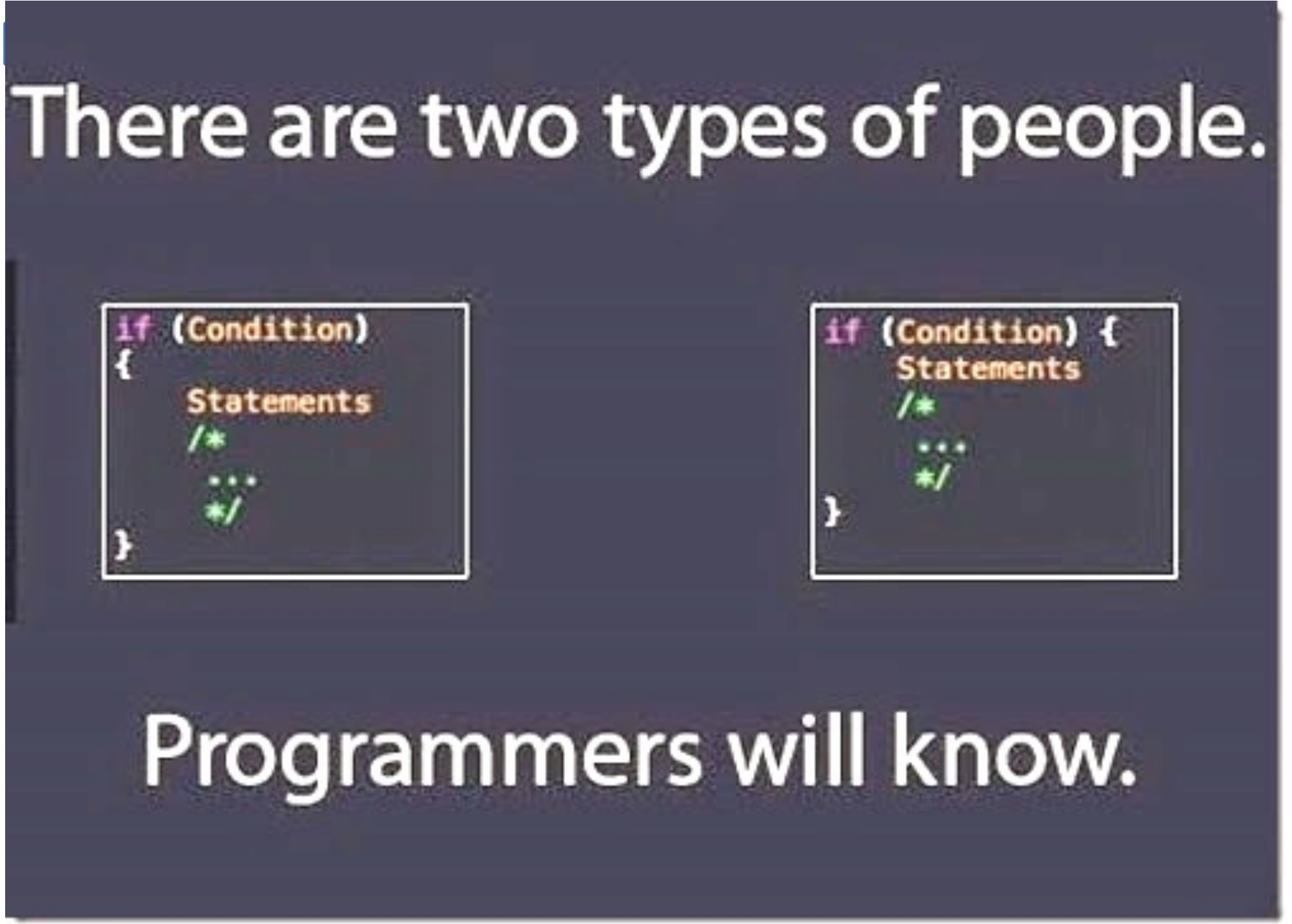
Conditional Execution

- Be careful about nested if statements
  - Consider `if(a > 0) if(b > 0) c = 5; else c = 6;`
  - Problem: the “else” block associated with which “if” ?
    - Rule: else joins with the innermost if
    - Can easily lead to confusion in human reading and interpretation
  - Always be unambiguous and explicit, by using braces

```
if(a > 0) {if(b > 0) c = 5; else c = 6;}

if(a > 0) {if(b > 0) c = 5;} else c = 6;
```

Conditional



Practice Examples for Lab: Set 5

- 1 done, swap

Write a program that reads 3 numbers and prints them in non-decreasing order.

- 2 done,

Write a program which takes as input a number denoting the `year`, and says whether the year is a leap year or not a leap year.

- 3 done

Write a program that takes as input 3 numbers a, b, c and prints out the roots of the quadratic equation $ax^2 + bx + c = 0$. Make sure that you handle all possible values of a, b, c without running into a division by zero or having to take the square root of a negative number. Even if the roots are complex, you should print them out suitably.

- 4 done

Write a program that reads in 3 characters. If the three characters consist of two digits with a '.' between them, then your program should print the square of the decimal number represented by the characters. Otherwise your program should print a message saying that the input given is invalid.

Practice Examples for Lab: Set 5

- 5 done

Write a program which prints all the prime numbers smaller than n, where n is to be read from the keyboard.

- 6 done

en.wikipedia.org/wiki/Perfect_number

A number is said to be perfect if it is equal to the sum of all numbers which are its factors (excluding itself). So for example, 6 is perfect, because it is the sum of its factors 1, 2, 3. Write a program which determines if a number is perfect. It should also print its factors.

Loops

- A motivating example

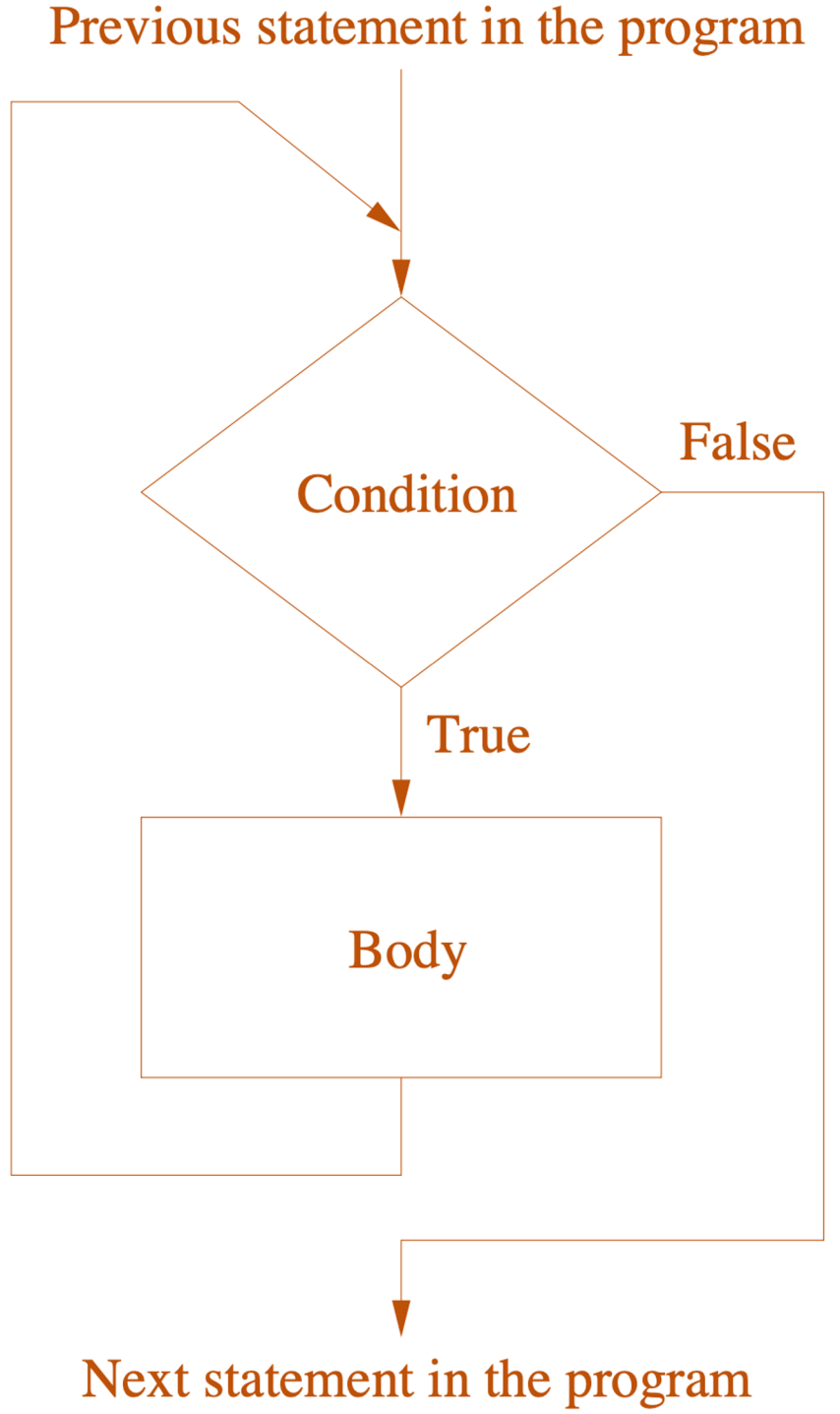
From the keyboard, read in a sequence of numbers, each denoting the marks obtained by students in a class. The marks are known to be in the range 0 to 100. The number of students is not told explicitly. If any negative number is entered, it is not to be considered the marks of any student, but merely a signal that all the marks have been entered. Upon reading a negative number, the program should print the average mark obtained by the students and stop.

- "Repeat" loop isn't great at solving this problem

```
repeat(count){
  statements
}
```

Loops

- Looping using “while”
  - while (condition) body
  1. Evaluate condition.
  2. If condition evaluates to true, then continue iterations, i.e., executing body once and return to step 1.
  3. If condition evaluates to false, then stop iterating.
- “body” needs to update some variable that has an effect on “condition”



Loops

- Any “repeat-count” loop can be replaced by an equivalent “while-condition” loop
  - unsigned int counter = 1;
    while (counter <= count)
    {
      statements;
      counter = counter + 1;
    }

```
repeat(count){
   statements
}
```

Loops

- While loop
  - Example: cubes of integers from 1 to 100

```
main_program{
   int i=1;
   while(i <= 100){
     cout << ‘‘The cube of ‘‘ << i << ‘‘ is ‘‘ << i*i*i << endl;
     i = i + 1;
   }
   cout << ‘‘Done!’’ << endl;
}
```

Loops

- While loop
  - Example: counting number of digits in an integer

```
main_program{
  int n;  cout << "Type a number: ";  cin >> n;

  int d = 1, ten_power_d=10;
          // ten_power_d will always be set to 10 raised to d

  while(n >= ten_power_d){  // if loop entered,
                            // number of digits in n must be > d
    d++;                    // so we try next choice for d
    ten_power_d *= 10;
  }

  cout << "The number has " << d << " digits." << endl;
}
```

Loops

- While loop
  - Example: averaging marks
    - Need to keep track of:
      (1) count of non-negative marks input so far
      (2) sum of non-negative marks input so far
    - Within each iteration, need to input marks and check if it is negative.
    - If input marks negative, then terminate loop; otherwise keep iterating
    - After exiting loop, compute average
    - Code assumes at least one non-negative mark entered (can modify to handle that case)

```
main_program{
  float nextmark, sum=0;
  int count=0;

  cin >> nextmark;

  while(nextmark >= 0){
    sum = sum + nextmark;
    count = count + 1;

    cin >> nextmark;
  }

  cout << "The average is: " << sum/count
}
```

Loops

- Example: averaging marks
- "break" inside body of a "while"
  - "condition" is always true
  - Must have a break inside body; otherwise we will have an "infinite loop"
    - Earlier version was easier to understand because terminating condition appeared at start of loop
    - Here, we need to search for a break statement inside body to figure out the logic
  - break gets control out of the (single) while within the body of which the break appeared
    - Be careful in case of nested while blocks

```
float nextmark, sum=0;
int count=0;

while(true){
   cin >> nextmark;
   if(nextmark < 0) break;
   sum = sum + nextmark;
   count = count + 1;
}
cout << sum/count << endl;
```

Loops

- What will this code do ?
  - int a = 0;
    while (a < 10)
    {
      cout << “a: “ << a << endl;
      if (a = 5) cout << “a equals 5” << endl;
      a++;
    }

Loops

- What will this code do ?
  - float x = 0.1;
    while (x != 1.1)
    {
      cout << “x: ” << x << endl;
      x += 0.1;
    }
  - Infinite loop can be caused by rounding errors in floating-point arithmetic
    - Some decimal numbers cannot be represented exactly in finite-length binary

C output on an AMD Turion processor:

x = 0.10000000149011611938

x = 0.20000000298023223877

x = 0.30000001192092895508

x = 0.40000000596046447754

x = 0.50000000000000000000

x = 0.60000002384185791016

x = 0.70000004768371582031

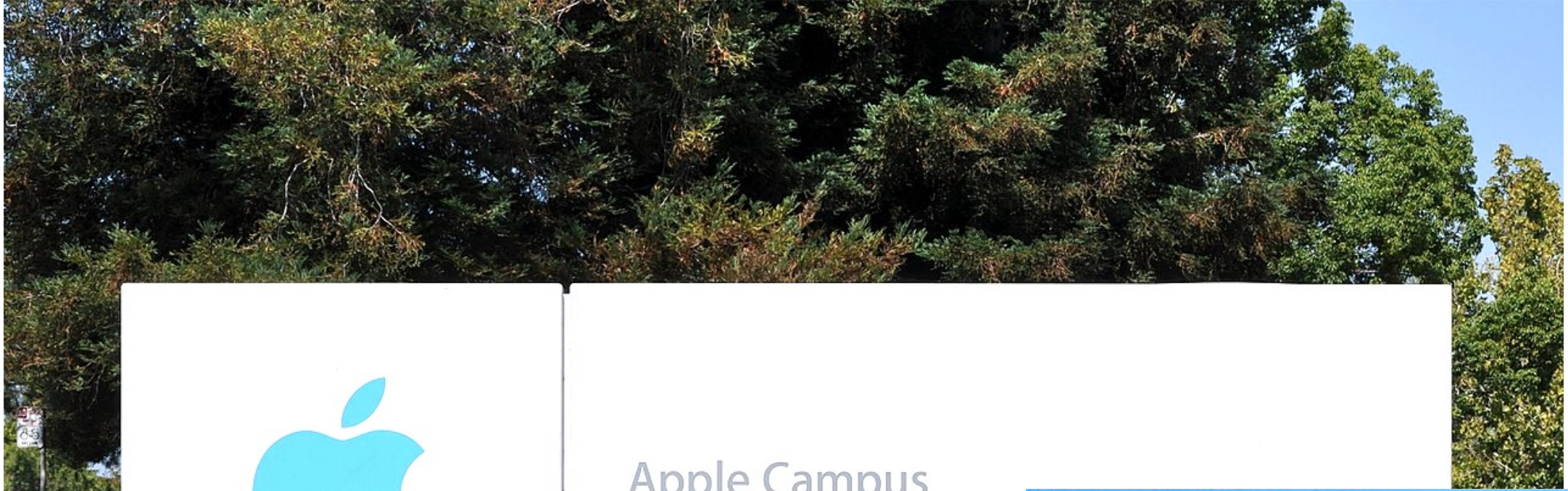
x = 0.80000007152557373047

x = 0.90000009536743164062

x = 1.00000011920928955078

x = 1.10000014305114746094

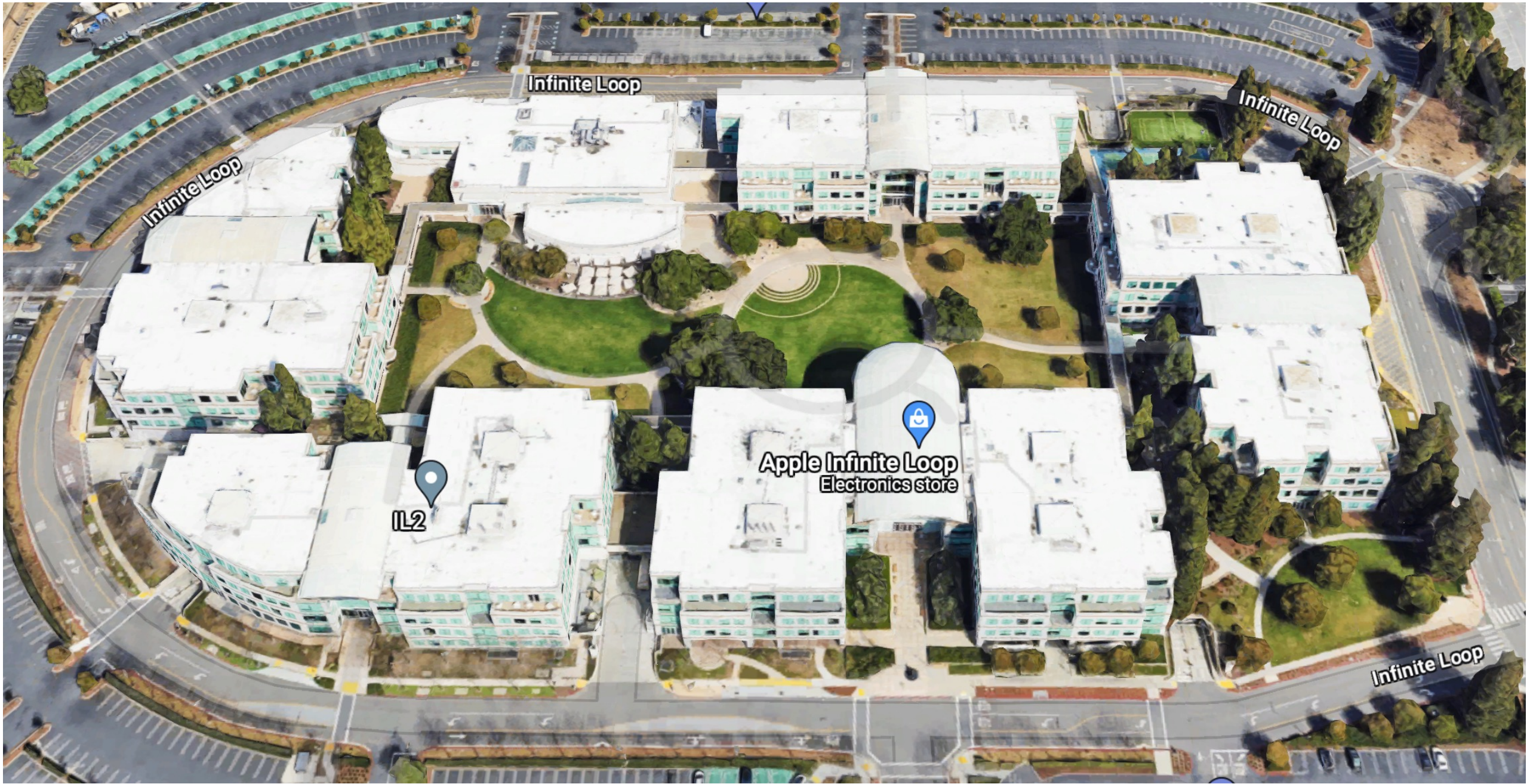
x = 1.20000016689300537109




Apple Campus
One Infinite Loop


INFINITE LOOP

Loops


Infinite Loop
Infinite Loop
Infinite Loop
Infinite Loop
IL2
Apple Infinite Loop
Electronics store

Loops

- "continue" inside body of a "while"
  - When a "continue" executes inside while loop, control goes back to start of loop

```
while(true){
   cin >> nextmark;
   if(nextmark > 100){
      cout << "Larger than 100, ignoring." << endl;
      continue;
   }
   if(nextmark < 0) break;
   sum = sum + nextmark;
   count = count + 1;
}
```

Loops

- do-while loop
  - do body
    while (condition);
    - Control flow enters loop
    - Execute body
    - Evaluate condition
    - If condition true, then control goes back to start of loop
    - If condition false, then control comes out of the loop
  - When we don't want to check terminating condition upon first entry into loop

```
main_program{
  float x;
  char response;

do{
    cout << ‘‘Type the number whose square root you want: ‘‘;
    cin >> x;
    cout << ‘‘The square root is: ‘‘ << sqrt(x) << endl;

    cout << ‘‘Type y to repeat: ‘‘;
    cin >> response;
    }
while(response == ’y’);
}
```

Loops

- "for" loop
  - Motivation:
    When execution inside body of loop relies on value of a (typically, int) variable, whose values across iterations follow a certain sequence
  - Instead of this:

```
int i = 1;
repeat(100){
  cout << i << ‘ ‘ << i*i*i << endl;
  i = i + 1;
}
```

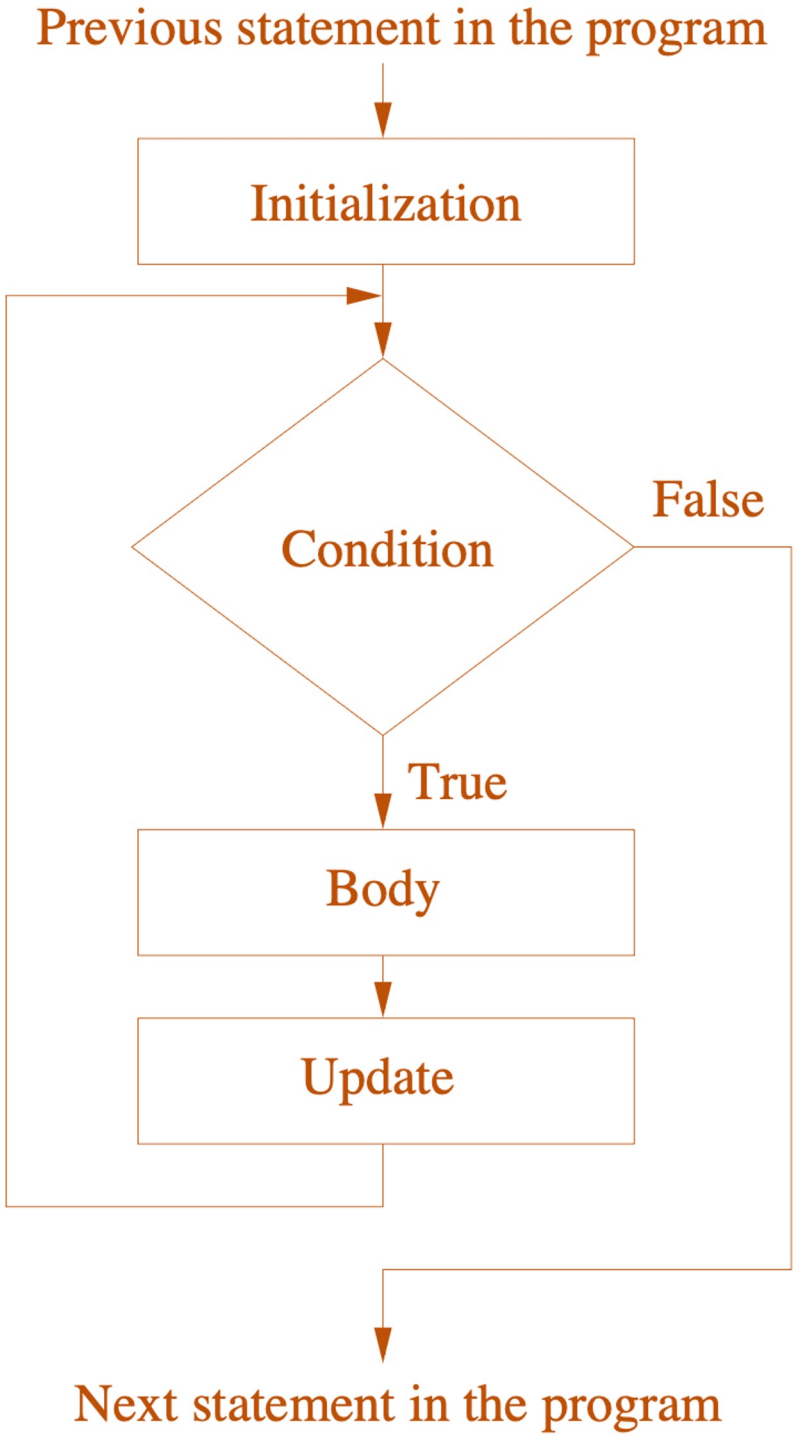
  - We do this:

```
for(int i=1; i <= 100; i = i + 1)
  cout << i << ‘ ‘ << i*i*i << endl;
```

  - Variable "i" called "control variable"
  - We iteratively execute body for different values of control variable i

Loops

- "for" loop
  - General form:
    for (initialization; condition; update) body
  - Execution:
    1. Execute initialization statement
    2. Evaluate condition.
    3. If false, then control goes out of loop
    4. If true then:
       - Execute body
       - Execute update statement
    5. Go back to Step 2
  - Special cases
    - Initialization statement can be empty
    - Update statement can be empty
    - Condition can be empty (then taken as true)



Loops

- “for” loop: scope of variables defined within “initialization”
  - Such variables:
    - Will get created within ‘for’ block
    - Will shadow other same-name variables defined earlier and in scope until ‘for’ block
    - Will get destroyed when control comes out of for block

```
int i=10;

for(int i=1; i<=100; i = i + 1) cout << i*i*i << endl;

cout << i << endl;
```

Loops

- “for” loop
  - for (initialization; condition; update) body
  - “break” statement inside body will terminate iterations; control comes out of for block
  - “continue” statement inside body will:
    - Immediately execute “update”
    - Return control to start of body

Loops

- “for” loop
  - Example of break statement inside for loop

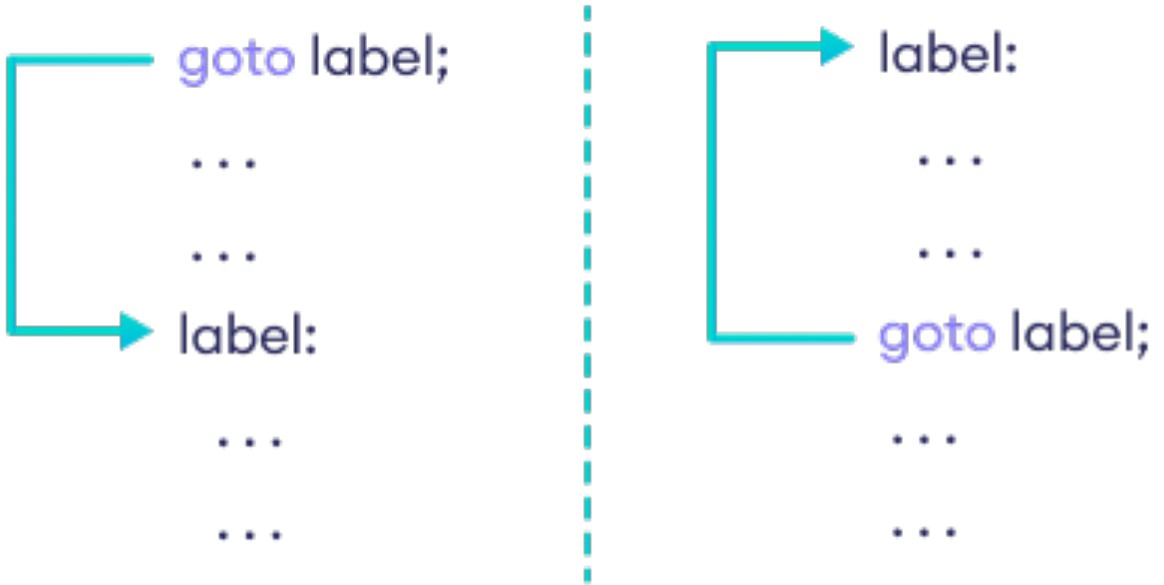
```
main_program{
   int x; cin >> x;

   bool found = false;
   for(int i=2; i < x; i++){
                                  // x is not divisible by 2...i-1
     if(x % i == 0){ found = true; break;}
   }

   if(found) cout << "Composite.\n";
   else cout << "Prime.\n";
}
```

Loops

- “break” and “continue” aren’t great coding practices
- Like “goto” statement in C/C++, in disguise
  - In C, one could “label” any statement and then have a “goto label” command to jump control to that statement



- Abrupt jumps in control flow make code more difficult to understand, maintain, debug
- Avoid as much as possible in while/for loops
  - One justified use of “break” is within switch statement
- Java programming language doesn’t have a goto statement

Practice Examples for Lab: Set 6

- 1 done

Write a program that prints a conversion table from Centigrade to Fahrenheit, say between 0° C to 100° C. Write using `while` and also using `for`.

- 2 done

Suppose we are given n points in the plane: $(x_1, y_1), \ldots, (x_n, y_n)$. Suppose the points are the vertices of a polygon, and are given in the counterclockwise direction around the polygon. Write a program using a `while` loop to calculate the perimeter of the polygon. Also do this using a `for` loop.

- 3 done

Write a program that returns the approximate square root of a non-negative integer. For this exercise define the approximate square root to be the largest integer smaller than the exact square root. Your are expected to not use the built-in `sqrt` or `pow` commands, of course. Your program is expected to do something simple, e.g. check integers in order $1, 2, 3, \ldots$ to see if it qualifies to be an approximate square root.

Practice Examples for Lab: Set 6

- 4 done

Write a program that prints out the digits of a number starting with the least significant digit, going on to the most significant. Note that the least significant digit of a number `n` is simply `n % 10`.

- 5 done

Write a program that takes a number `n` and prints out a number `m` which has the same digits as `m`, but in reverse order.

- 6 done

A natural number is said to be a palindrome if the sequence its digits is the same whether read left to right or right to left. Write a program to determine if a given number is a palindrome.

Goto

- Spaghetti code
  - en.wikipedia.org/wiki/Spaghetti_code
  - “Code has a complex and tangled control structure, resulting in a program flow that is conceptually like a bowl of spaghetti, twisted and tangled”
  - “Can be caused by several factors, such as volatile project requirements, lack of programming style rules, and software engineers with insufficient ability or experience.”

```
#include <stdio.h>

int main()
{
    puts("This is Line 1");
    goto this;
that:
    puts("This is Line 3");
    goto theother;
backhere:
    puts("This is Line 5");
    goto end;
this:
    puts("This is Line 2");
    goto that;
theother:
    puts("This is Line 4");
    goto backhere;
end:

    return(0);
}
```

Goto

- The C Programming Language – Kernighan and Ritchie

3.8 Goto and Labels

C provides the infinitely-abusable `goto` statement, and labels to branch to. Formally, the `goto` is never necessary, and in practice it is almost always easy to write code without it. We have not used `goto` in this book.

Nevertheless, there are a few situations where `goto`s may find a place. The most common is to abandon processing in some deeply nested structure, such as breaking out of two or more loops at once. The `break` statement cannot be used directly since it only exits from the innermost loop. Thus:

```
for ( ... )
    for ( ... ) {
        ...
        if (disaster)
            goto error;
    }
...

error:
    clean up the mess
```

Goto

dl.acm.org/doi/10.1145/362929.362947

ARTICLE FREE ACCESS

Letters to the editor: go to statement considered harmful

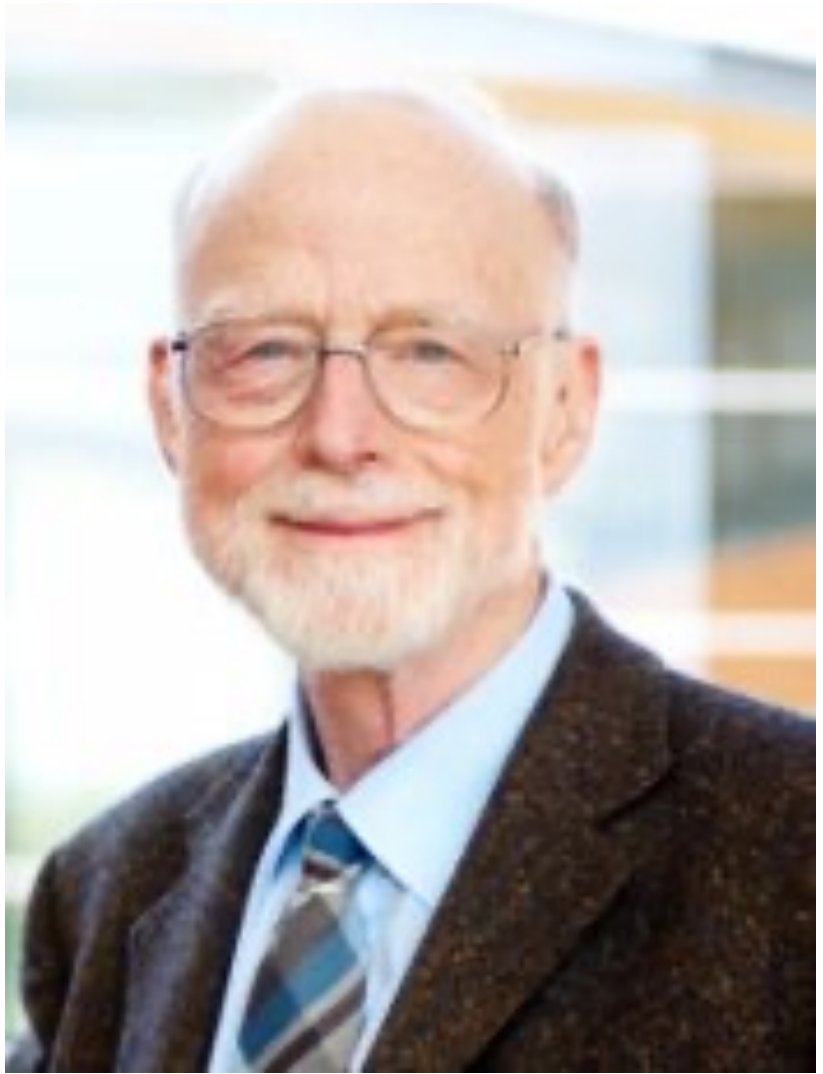
Author:  Edsger W. Dijkstra Authors Info & Claims

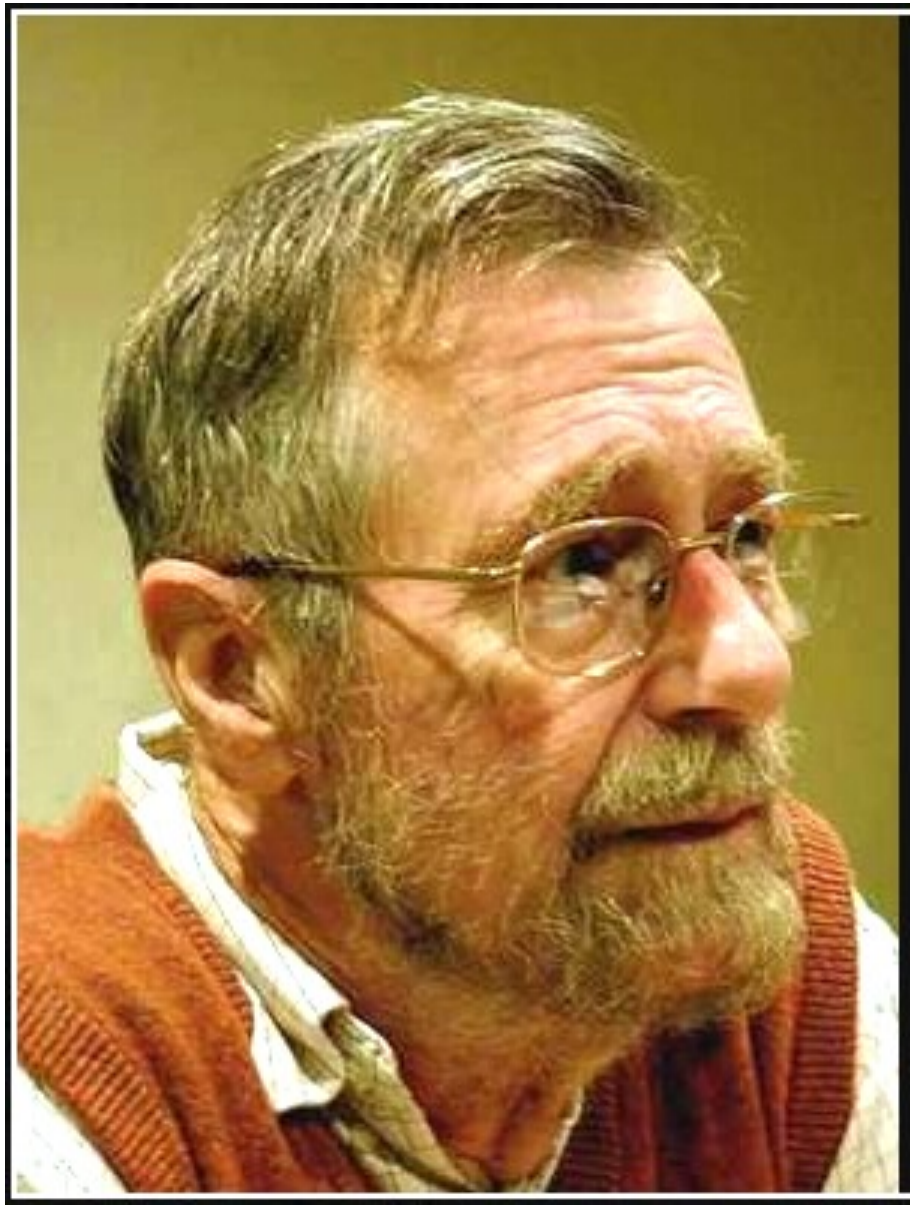
Communications of the ACM, Volume 11, Issue 3 • 01 March 1968 • pp 147–148 • https://doi.org/10.1145/362929.362947

Published: 01 March 1968 Publication History

Goto

- Edsger W. Dijkstra
- Dutch computer scientist, programmer, software engineer, systems scientist, and science essayist
- 1972 Turing Award
  for fundamental contributions to
  developing structured programming languages
- PhD thesis (1959) title:
  Communication with an Automatic Computer





Program testing can be used to show the presence of bugs, but never to show their absence!

— Edsger Dijkstra —

AZ QUOTES

Turing Award

- Annual prize given by Association for Computing Machinery (ACM) for contributions of lasting and major technical importance to computer science
  - en.wikipedia.org/wiki/Turing_Award
  - Highest distinction in computer science
  - “Nobel Prize of Computing”
- Named after Alan Turing
  - en.wikipedia.org/wiki/Alan_Turing
  - British mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist
  - Father of theoretical computer science and artificial intelligence

Association for Computing Machinery (ACM)

- US-based international learned society for computing
- en.wikipedia.org/wiki/Association_for_Computing_Machinery
- Founded in 1947
- World's largest scientific and educational computing society
- Several journals published by ACM
  - *ACM Transactions on Algorithms* (TALG)
  - *ACM Transactions on Embedded Computing Systems* (TECS)
  - *ACM Transactions on Computer Systems* (TOCS)
  - *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (TCBB)
  - *ACM Transactions on Computational Logic* (TOCL)
  - *ACM Transactions on Computer-Human Interaction* (TOCHI)
  - *ACM Transactions on Database Systems* (TODS)
  - *ACM Transactions on Graphics* (TOG)
  - *ACM Transactions on Mathematical Software* (TOMS)
  - *ACM Transactions on Multimedia Computing, Communications, and Applications* (TOMM)
  - *IEEE/ACM Transactions on Networking* (TON)
  - *ACM Transactions on Programming Languages and Systems* (TOPLAS)

Goto

- In 1974, at the peak of an international debate that followed Dijkstra's article, "Go To Statement Considered Harmful," Knuth wrote,

1. ELIMINATION OF go to STATEMENTS

Historical Background

At the IFIP Congress in 1971 I had the pleasure of meeting Dr. Eiichi Goto of Japan, who cheerfully complained that he was always being eliminated. Here is the history of the subject, as far as I have been able to trace it.

  - dl.acm.org/doi/pdf/10.1145/356635.356640

- Eiichi Goto
  - Japanese computer scientist, the builder of one of the first general-purpose computers in Japan

Goto

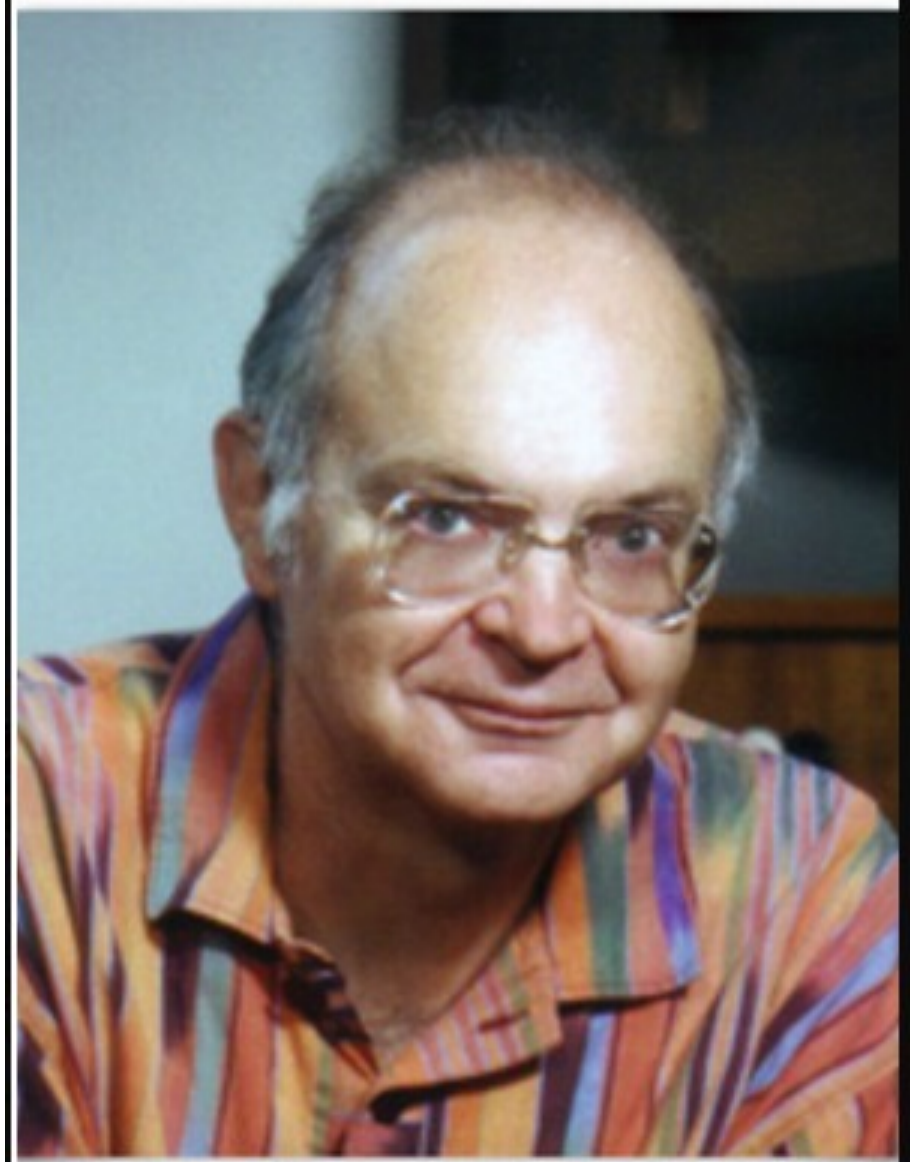
- Donald Knuth
  - American computer scientist, mathematician
  - 1974 recipient of the ACM Turing Award
  - “Father of the analysis of algorithms”
  - Author of “The Art of Computer Programming”
    - Opus in progress over 50 years

Completed [edit]

- Volume 1 – Fundamental Algorithms
  - Chapter 1 – Basic concepts
  - Chapter 2 – Information structures
- Volume 2 – Seminumerical Algorithms
  - Chapter 3 – Random numbers
  - Chapter 4 – Arithmetic
- Volume 3 – Sorting and Searching
  - Chapter 5 – Sorting
  - Chapter 6 – Searching
- Volume 4A – Combinatorial Algorithms
  - Chapter 7 – Combinatorial searching (part 1)
- Volume 4B – Combinatorial Algorithms
  - Chapter 7 – Combinatorial searching (part 2)

Planned [edit]

- Volume 4C... – Combinatorial Algorithms (chapters 7 & 8 released in several subvolumes)
  - Chapter 7 – Combinatorial searching (continued)
  - Chapter 8 – Recursion
- Volume 5 – Syntactic Algorithms
  - Chapter 9 – Lexical scanning (also includes string search and data compression)
  - Chapter 10 – Parsing techniques
- Volume 6 – The Theory of Context-Free Languages
- Volume 7 – Compiler Techniques

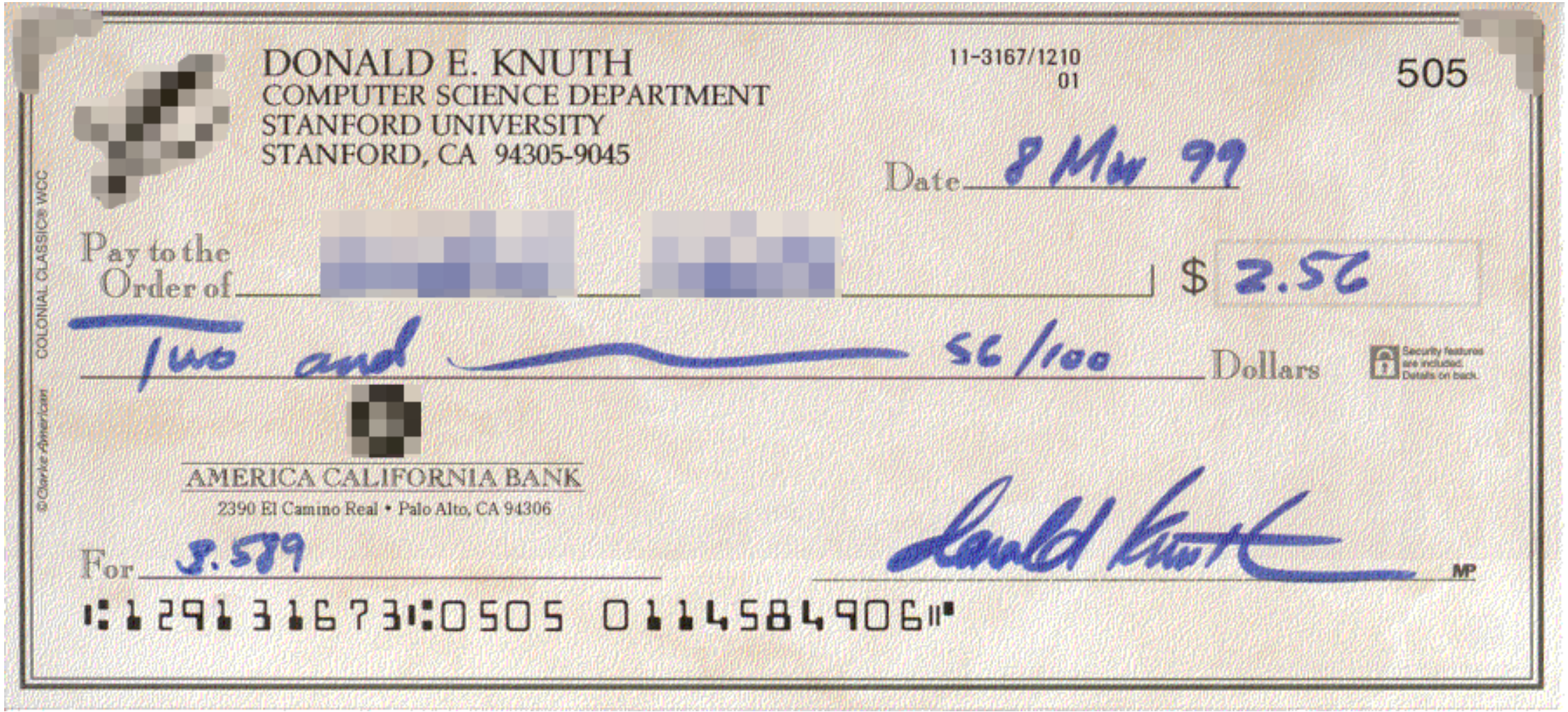


Computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better.

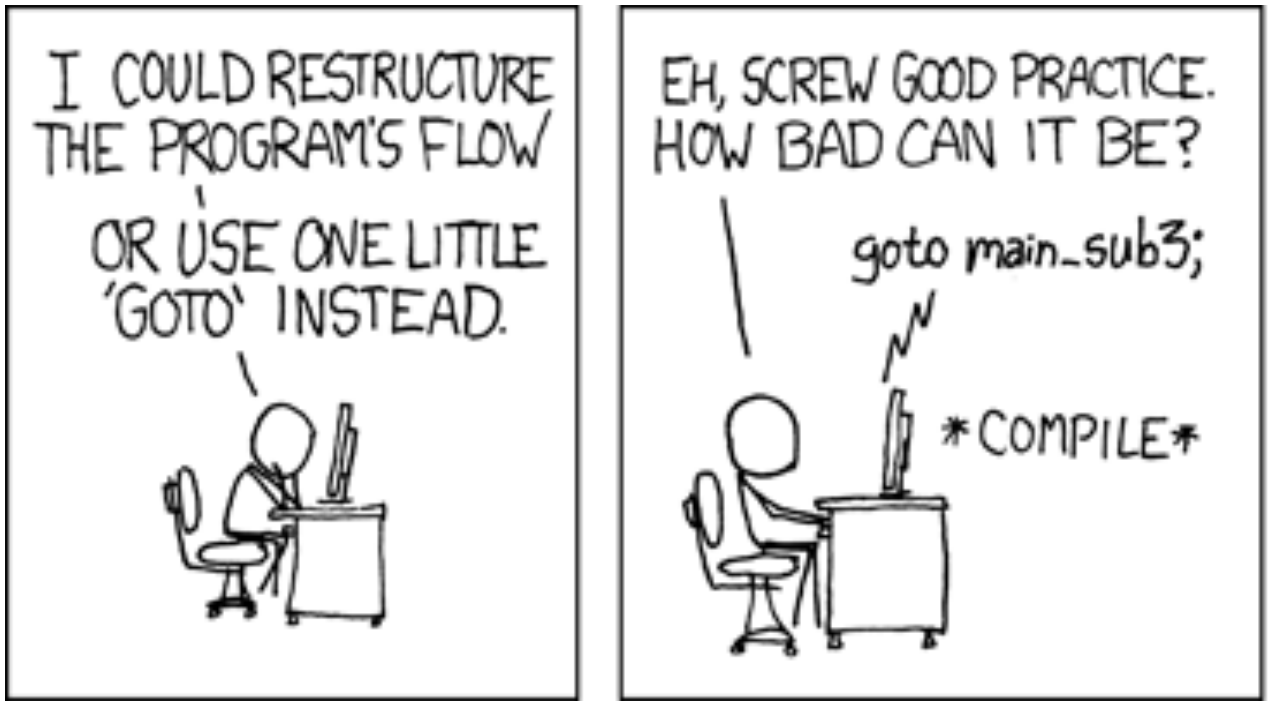
— Donald Knuth —

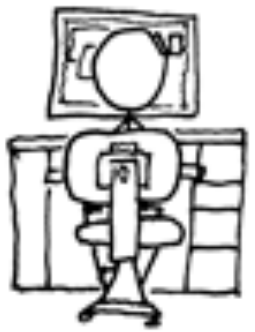
AZ QUOTES

- en.wikipedia.org/wiki/Knuth_reward_check
  - “The MIT Technology Review describes the checks as "among computerdom's most prized trophies”



Goto


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE



The pasta theory of programming

Spaghetti code

Unstructured and hard-to-maintain code caused by lack of style rules or volatile requirements. This architecture resembles a tangled pile of spaghetti in a bowl.

Lasagna code

Source code with overlapping layers, like the stacked design of lasagna. This code structure makes it difficult to change one layer without affecting others.

Ravioli code

Isolated bits of code that resemble ravioli. These are easy to understand individually but—taken as a group—add to the app's call stack and complexity.

Pizza code

A codebase with interconnected classes or functions with unclear roles or responsibilities. These choices result in a flat architecture, like toppings on a pizza.

Loops

- "for" loop (possibilities/tricks)
  - "Initialization" and "update" can have multiple assignment statements separated by commas
  - Example of digit-counting problem

To find num of digits.

```
main_program{
  int n; cin >> n;

  int d, ten_power_d;
  for(d=1, ten_power_d = 10; ten_power_d <= n; d++, ten_power_d *= 10);

  cout << "The number has " << d << " digits." << endl;
}
```

    - This for statement doesn't have a body
      - For this example, code is compact (and clever), but for loop without body isn't good coding style

Loops

Important slides.

- "for" loop (possibilities/tricks)
  - "Initialization" and "update" can have input statements via cin
  - Example of mark-averaging problem

```
main_program{
  float nextmark,sum=0;
  float count=0;

  for(cin >> nextmark; nextmark >= 0; cin >> nextmark){
    count++;
    sum += nextmark;
  }
  cout << sum/count;
}
```

  - Tricks usually come at the cost of readability

Loops

- Greatest common divisor (GCD)
  - Consider 2 positive integers: m,n with m > n
  - Many possible algorithms for computing GCD:
    - Check for all numbers between 1 and min(m,n); choose largest that divides both
    - Find prime factorizations; find common factors; find their product
    - Euclid's algorithm, based on the following principle:
      D divides m and n if and only if D divides m-n and n.
      Thus (when m > n) : GCD(m,n) = GCD(m-n,n)
      - Reduce problem to finding GCD of (m-n) and (n). Proof is straightforward.
    - Example
      - GCD (3977, 943)
      - = GCD (3977 - 943, 943) = GCD (3034, 943)
      - = GCD (3034 - 943, 943) = GCD (2091, 943)
      - = GCD (2091 - 943, 943) = GCD (1148, 943)
      - = GCD (1148 - 943, 943) = GCD (205, 943)
      - Observation: we subtracted 943 multiple times until result was <= 943

Loops

- Greatest common divisor (GCD)
  - Modified Euclid's principle:
    D divides m and n if and only if D divides m%n and n.
    Thus (when m % n > 0) : GCD(m,n) = GCD(m%n,n)
  - Example:

	GCD (3977, 943)		GCD(m,n)
= GCD (3977 % 943, 943)	= GCD (205, 943)	= GCD (943, 205)	GCD (m',n')
= GCD (943 % 205, 205)	= GCD (123, 205)	= GCD (205, 123)	GCD (m'',n'')
= GCD (205 % 123, 123)	= GCD (82, 123)	= GCD (123, 82)	
= GCD (123 % 82, 82)	= GCD (41, 82)	= GCD (82, 41)	
= GCD (82 % 41, 41)	→ but 82 % 41 is zero		

  - Observation: stopping criterion should be when modulo operation produces 0; then divisor is the answer

At the time n divides m completely the GCD is found.

Theorem 1 (Euclid) *Suppose m, n are positive integers. If $m\%n = 0$, then $GCD(m, n) = n$. Otherwise $GCD(m, n) = GCD(m\%n, n)$.*

Loops

Theorem 1 (Euclid) *Suppose m, n are positive integers. If $m\%n = 0$, then $GCD(m, n) = n$. Otherwise $GCD(m, n) = GCD(m\%n, n)$.*

- Greatest common divisor (GCD)
  - Modified Euclid's algorithm

```
main_program{ // Compute GCD of m,n, where m > n >0.
  int m,n;
  cout << "Enter the larger number (must be > 0): ";  cin >> m;
  cout << "Enter the smaller number (must be > 0): "; cin >> n;

  while(m % n != 0){
    int Remainder = m % n;
    m = n;
    n = Remainder;
  }
  cout << "The GCD is: " << n << endl;
}
```

Loops

Theorem 1 (Euclid) *Suppose m, n are positive integers. If $m\%n = 0$, then $GCD(m, n) = n$. Otherwise $GCD(m, n) = GCD(m\%n, n)$.*

- Greatest common divisor (GCD)
  - Modified Euclid's algorithm: Analysis
  - Loop invariants
    - By Theorem 1, before and after each iteration, GCD remains unchanged
    - As per code, before and after each iteration, we have m > n > 0
  - Is termination guaranteed ?
    - At start of some iteration, if m % n = 0, then loop body isn't entered; we terminate.
    - If loop body entered
      - During each iteration, m % n always < n, and positive
      - Thus, after each iteration, max(m,n) has reduced (because m>n at start)
      - So, we will need at most max(m,n) iterations to reach termination
      - Actually it is much less:
      - Doing m % n gives m = Qn + R >= n + R (because Q >= 1; because m > n). Re-assign: m' = n and n' = R
      - [if enter body again] Doing m' % n' gives m' >= n' + R' > n'. So n > R. Re-assign: m'' = n' = R and …
      - Thus, m >= n + R = n + m'' = m' + m'' >= m'' + m'' = 2m'' (thus, in 2 iterations, m at least halved)

Computing Mathematical Functions

Finding the value of a fn using tailer theorem

- Taylor series
  - In general, if x is reasonably close to x_0, then

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2!} + f'''(x_0)\frac{(x - x_0)^3}{3!} + \cdots$$

    and computing first few (low-order) terms of series gives a good estimate
    - For some functions (e.g., sqrt, log, tan, arctan) it doesn't converge as x goes far from x_0
    - For some functions, (e.g., exp, sin, cos) it converges everywhere
    - en.wikipedia.org/wiki/Taylor_series#Analytic_functions
  - Another way of expressing Taylor series (define h := x - x_0)

$$f(x_0 + h) = f(x_0) + f'(x_0)h + f''(x_0)\frac{h^2}{2!} + f'''(x_0)\frac{h^3}{3!} + \cdots$$

  - Choosing x_0 = 0 gives McLaurin series

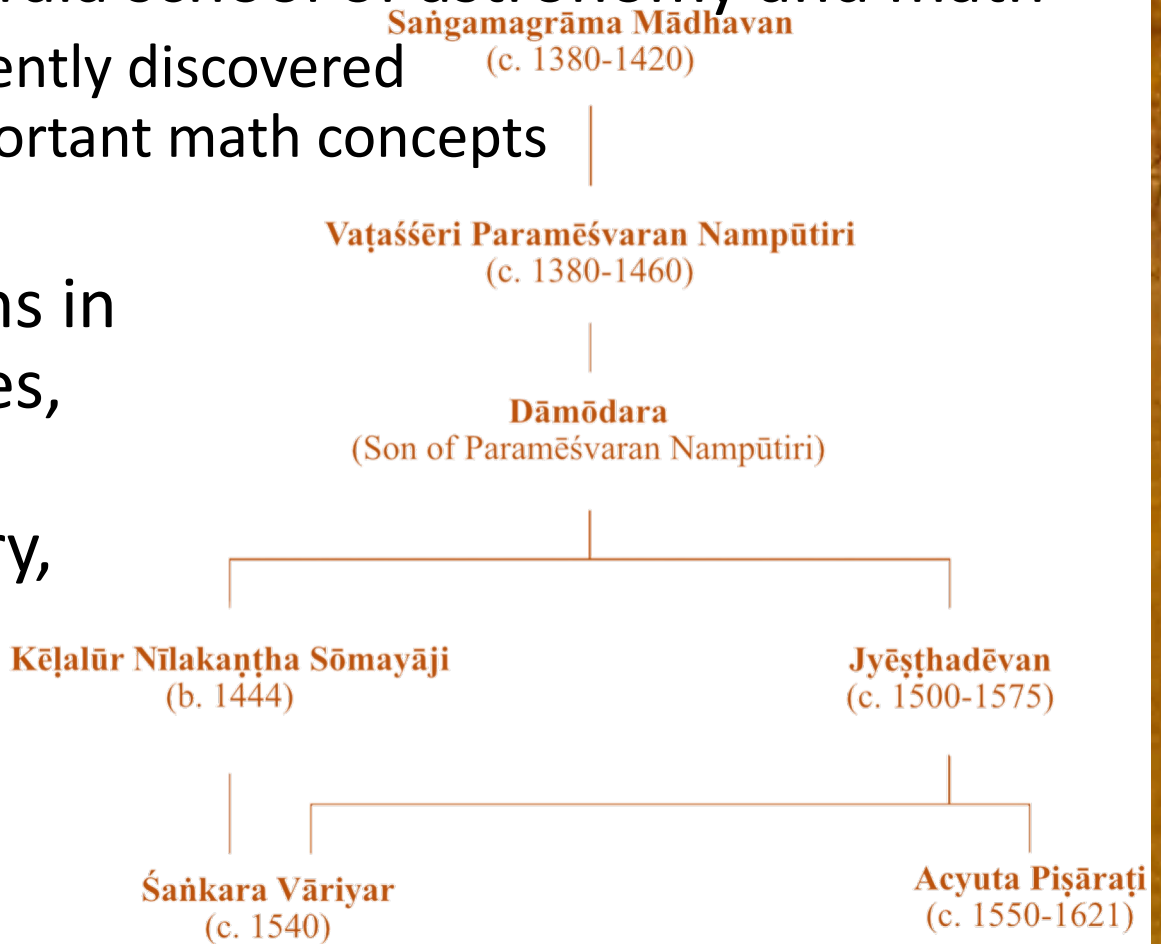
$$f(x) = f(0) + f'(0)x + f''(0)\frac{x^2}{2!} + f'''(0)\frac{x^3}{3!} + \cdots$$

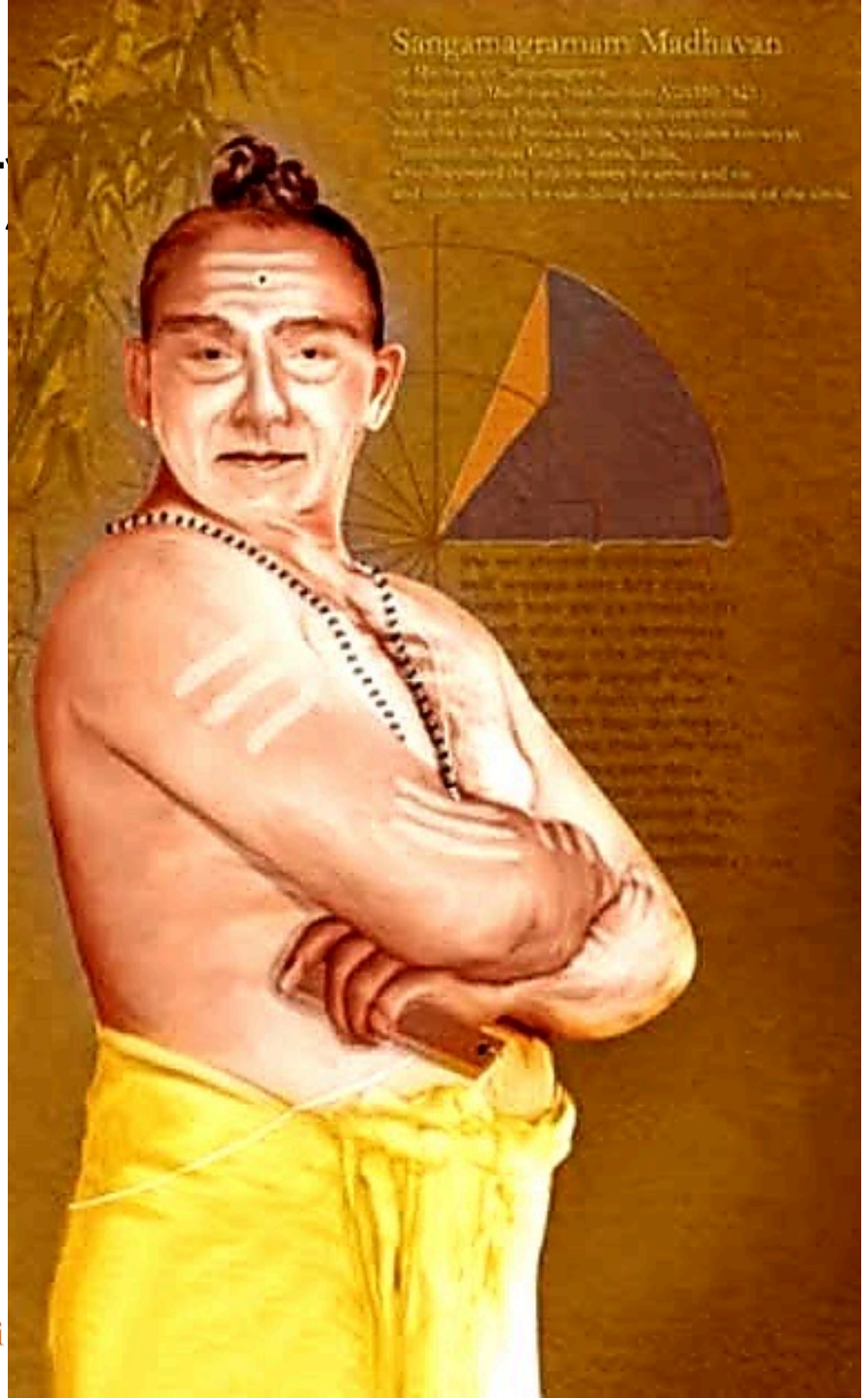
Computing Mathematical Functions

- Brook Taylor
  - British; first studied law at Cambridge Univ.: LL.B. and LL.D.
  - Proposed Taylor's theorem in 1715, which remained unrecognized until 1772, when Lagrange called it "main foundation of differential calculus"
  - On committee adjudicating some works of Newton & Leibniz
- Collin McLaurin
  - Scottish; child prodigy; entered U. Glasgow at age 11
  - Age 19 (year 1717): elected math professor at U Glasgow; was a world record for youngest prof, broken only in 2008
  - Joined U Edinburgh in 1725, where Newton, very impressed, offered to pay McLaurin's salary himself
  - McLaurin's series known before Taylor; some special cases relate to Madhava of Sangamagrama in 14th century India

Computing Mathematical Functions

- Mādhava (माधव) of Sangamagrāma (संगमग्राम)
  - One of greatest mathematician-astronomers of Middle Ages
  - Founded Kerala school of astronomy and math
    - Independently discovered many important math concepts
  - Pioneering contributions in infinite series, calculus, trigonometry, algebra, geometry, value of pi





Computing Mathematical Functions

- Choosing f(.) as sin(.) function (with angle in radians) and $x_0 = 0$ gives:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

  - To evaluate sin(x) for $x > 2\pi$, we evaluate sin(x % 2π) instead to keep x small

$$t_k = (-1)^{k+1}\frac{x^{2k-1}}{(2k-1)!} = t_{k-1}\left((-1)\frac{x^2}{(2k-2)(2k-1)}\right)$$

```
main_program{
  double x;  cin >> x;

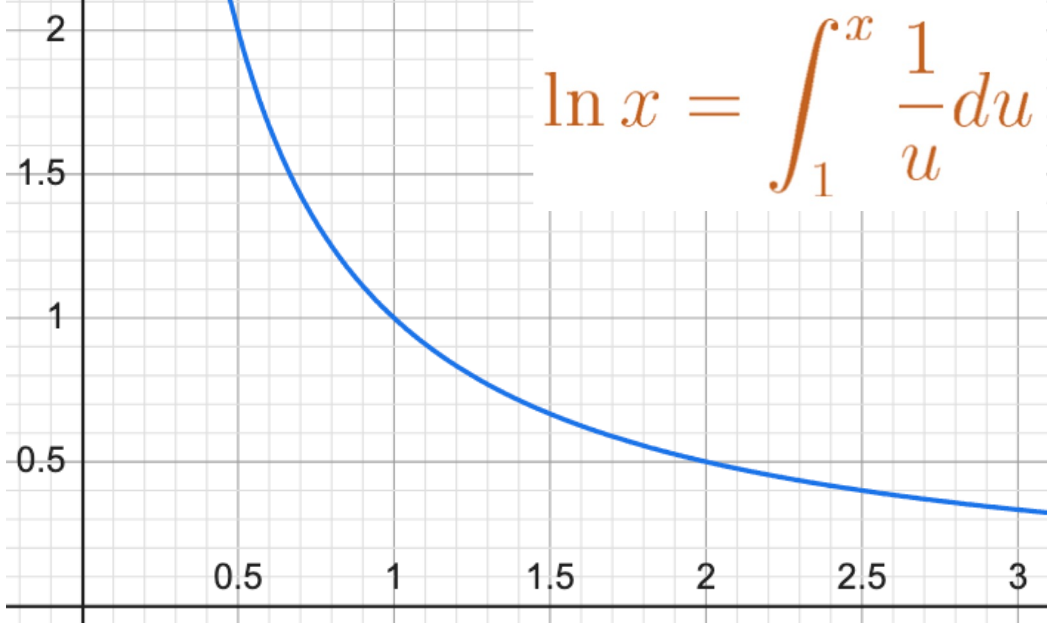
  double epsilon = 1.0E-20, sum = x, term = x;

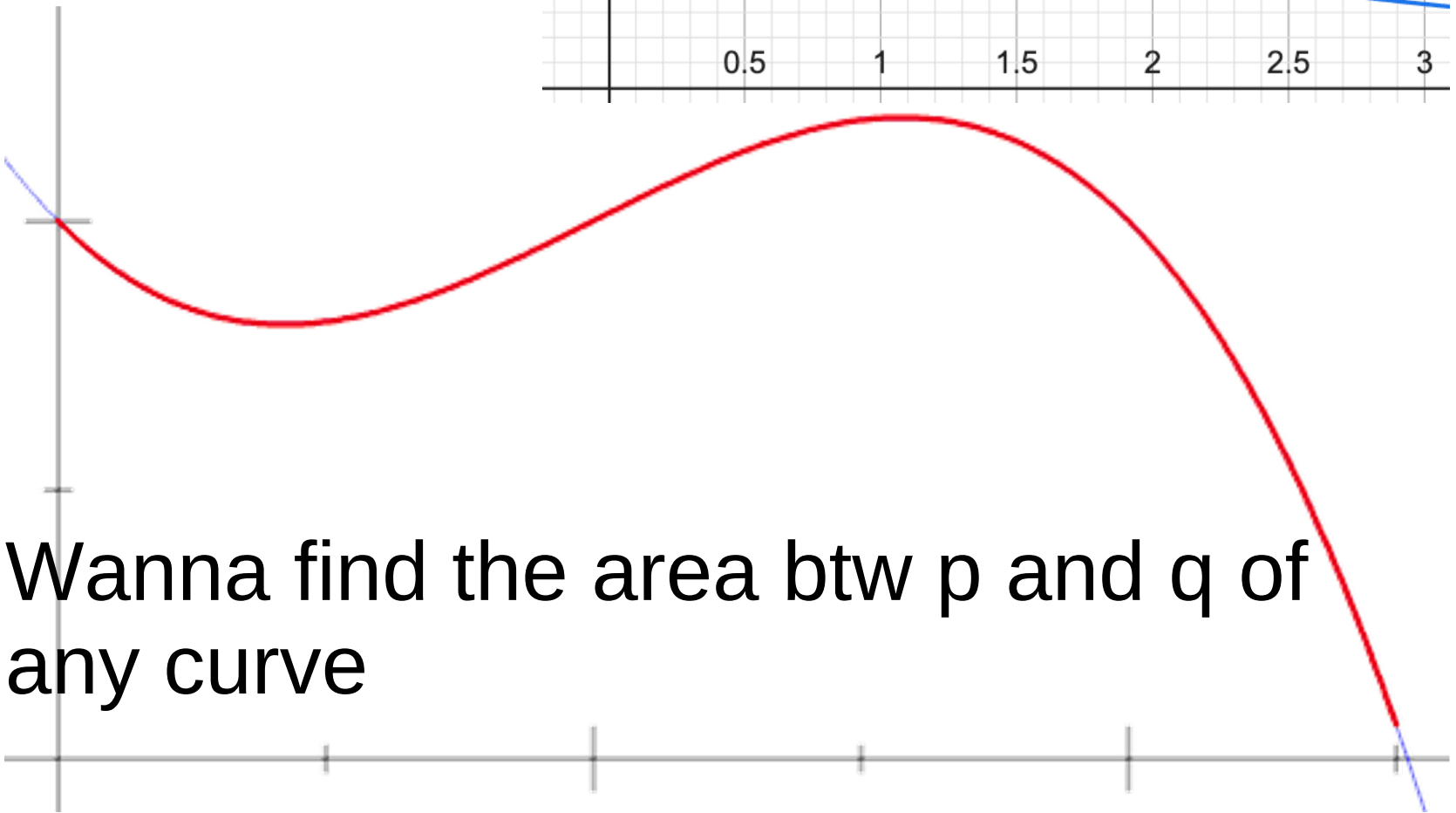
  for(int k=2; abs(term) > epsilon; k++){
                // Plan: term = t_{k-1}, sum = sum of k-1 terms
    term *= -x * x /((2*k-2)*(2*k-1));
    sum += term;
  }

  cout << sum << endl;
}
```

Computing Mathematical Functio

- Numerical Integration
  - ln(.) = area under part of 1/u curve between [1,x]
  - Riemann integration
  - Kinds of errors in numerical integration
    - Due to theoretical approximation
    - Due to approximation in floating-point calculation: 4-byte-floating-point representation is correct only up to 7 decimal places

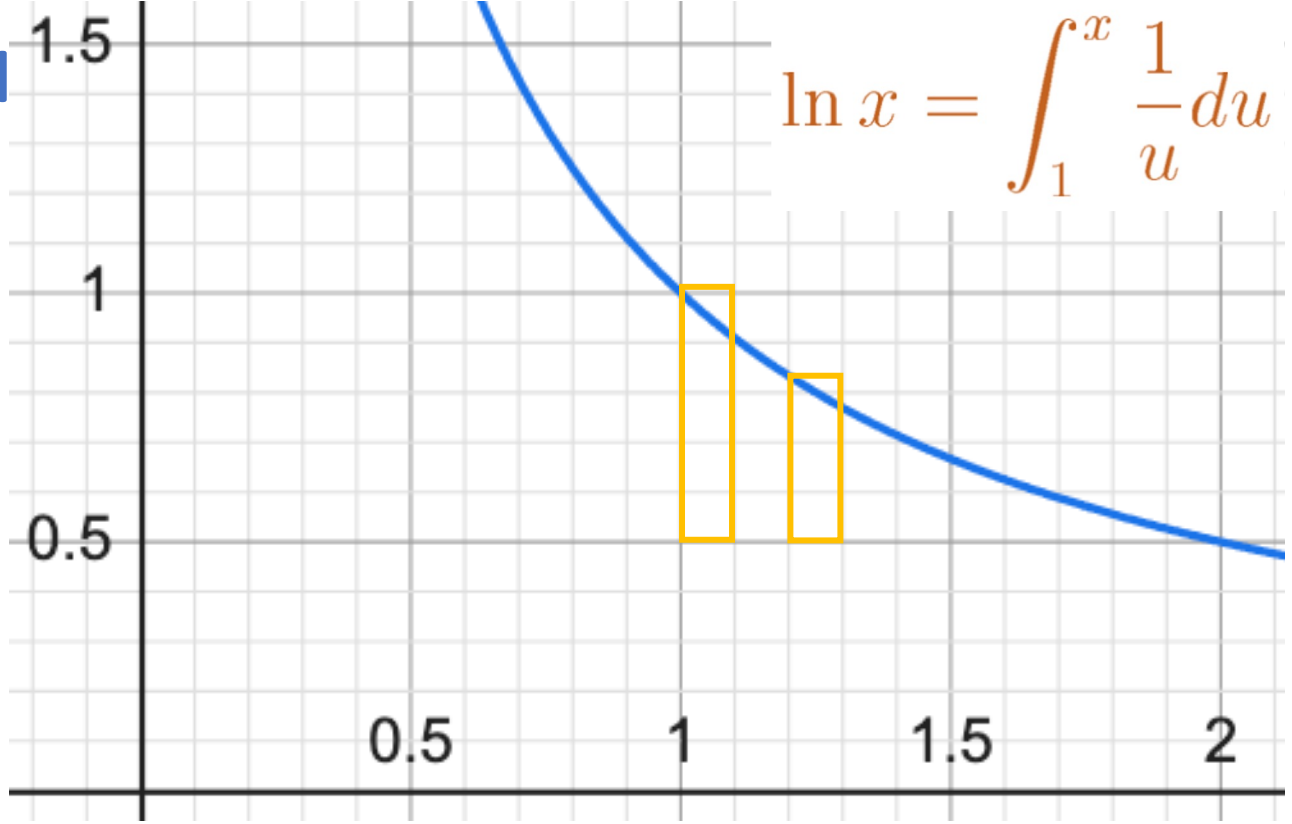




Computing Mathematical Fu

- Numerical Integration
  - Divide into n vertical strips
  - Strip width = w = (x-1)/n
  - i-th strip extends from u=1+iw to u=1+(i+1)w, where i takes values 0,1,…,n-1
  - i-th strip's height = 1/(1+iw); overestimate
  - Sum of areas of all strips =

$$\sum_{i=0}^{n-1} w \frac{1}{1+iw}$$

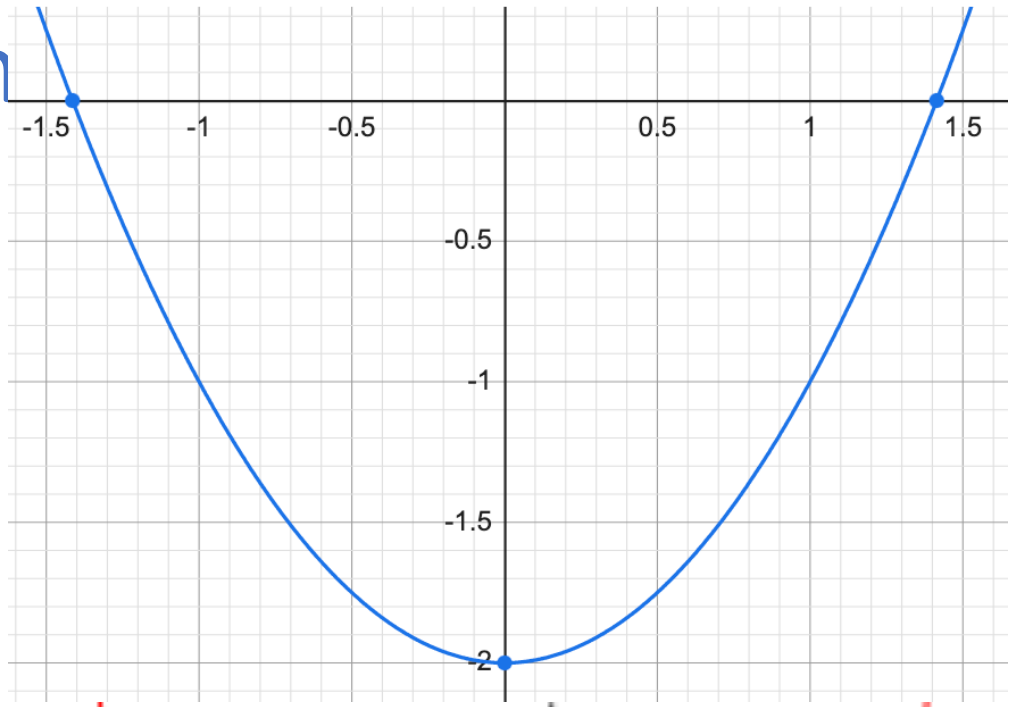


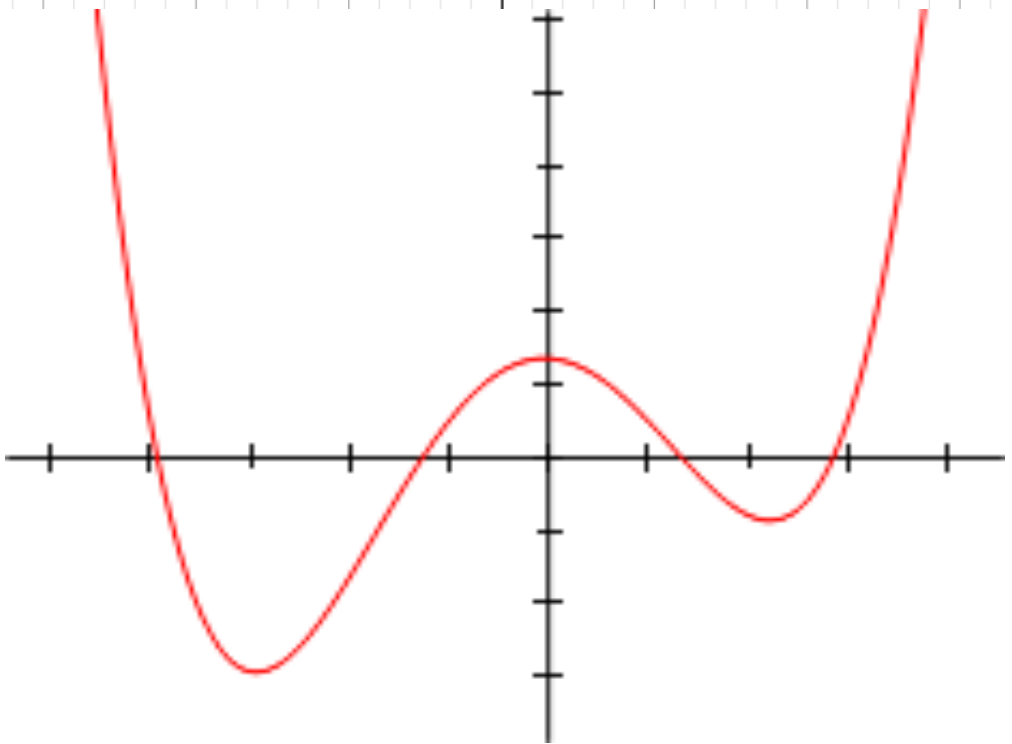

```
main_program{
   float x; cin >> x;          // will calculate ln(x)
   int n; cin >> n;            // number of rectangles to use
   float w = (x-1)/n;          // width of each rectangle
   float area = 0;             // will contain ln(x) at the end.
   for(int i=0; i < n; i++)
      area = area + w /(1+i*w);
   cout << "Natural log, from integral: "<< area << endl;
}
```

Riemann Integration.

Computing Mathematical Function

- Computing a root of a function
- For a function f(x),
  a root is defined as
  a value x' for which f(x')=0
- Example
  - Can use this to find roots of numbers, i.e., sqrt(2) equals root of function $f(x)=x^2-2$
  - Can use this to find
    local maximum/minimum of a function
    - Finding x that minimizes f(x) can be framed as first finding root of g(x) = df(x)/dx
      - Then checking for sign of $d^2f(x)/d^2x$

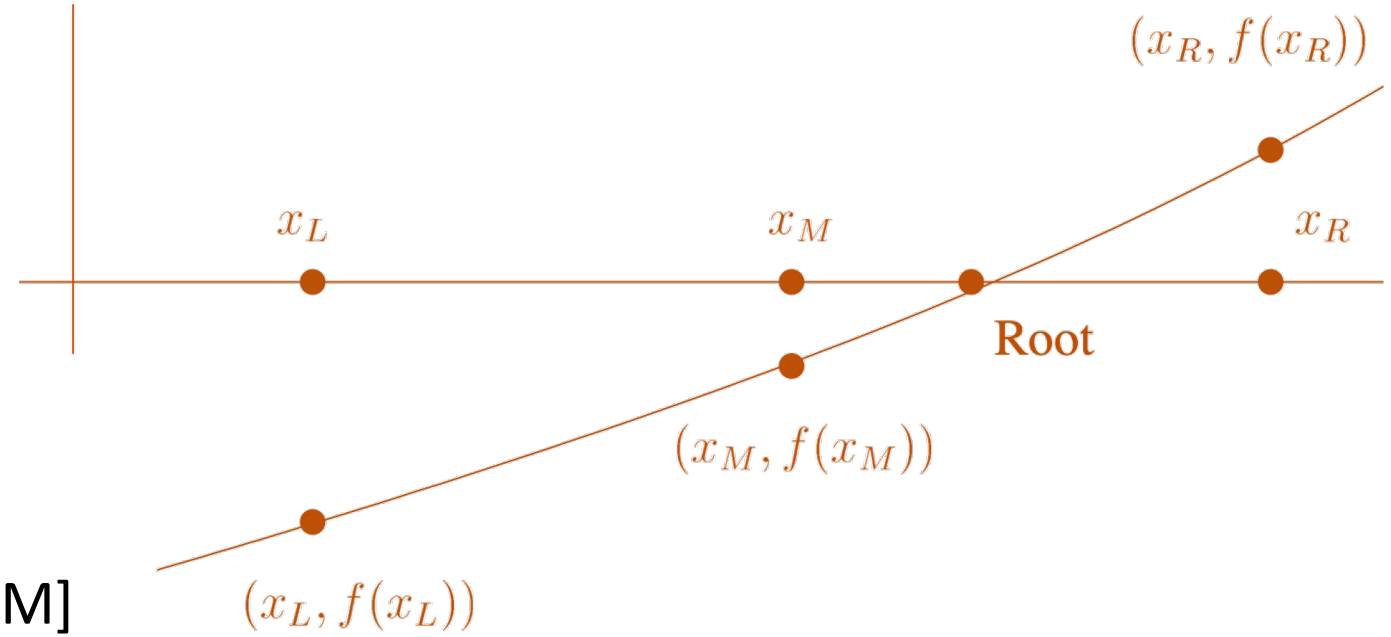




Computing Mathematical Functions

It is find root of any function.

- Bisection method for computing function root
- Assumptions
  - Given: values xL & xR such that: xL < xR, and f(xL) & f(xR) have opposite signs
  - f(x) is continuous within interval [xL,xR]
- Algorithm
  - Consider interval's mid-point
    xM = (xL + xR) / 2
  - Evaluate sign of f(xM)
  - If sign(f(xL)) same as sign(f(xM)), then a root exists within [xM,xR]
    - Else, there a root exists within [xL,xM]
  - Redefine interval and repeat
  - When do we stop ? Should we check for f(xM) == 0 ?

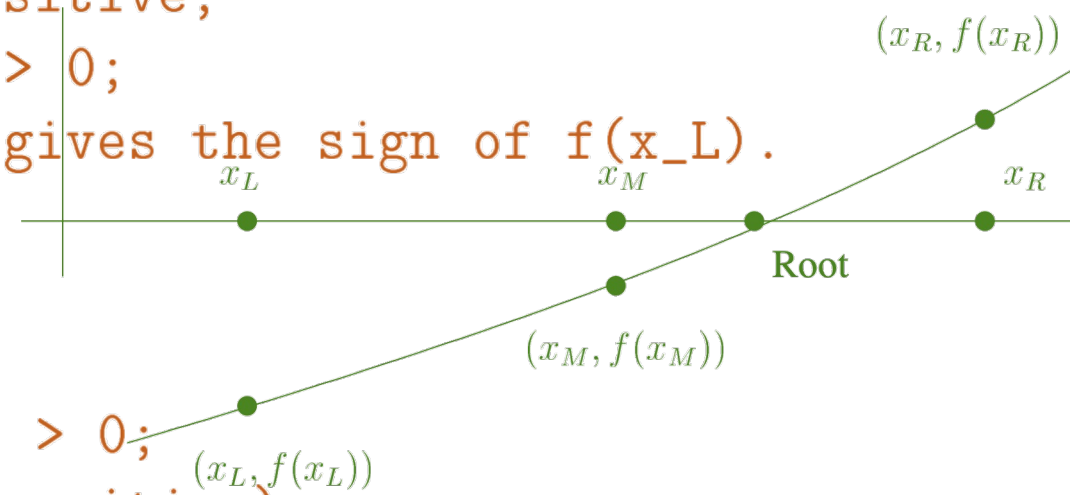


Computing Mathematical Functions

- Bisection method for computing function root
  - Is this code correct ?
- Analysis
  - Will loop terminate ?
  - Are we improving each iteration ?

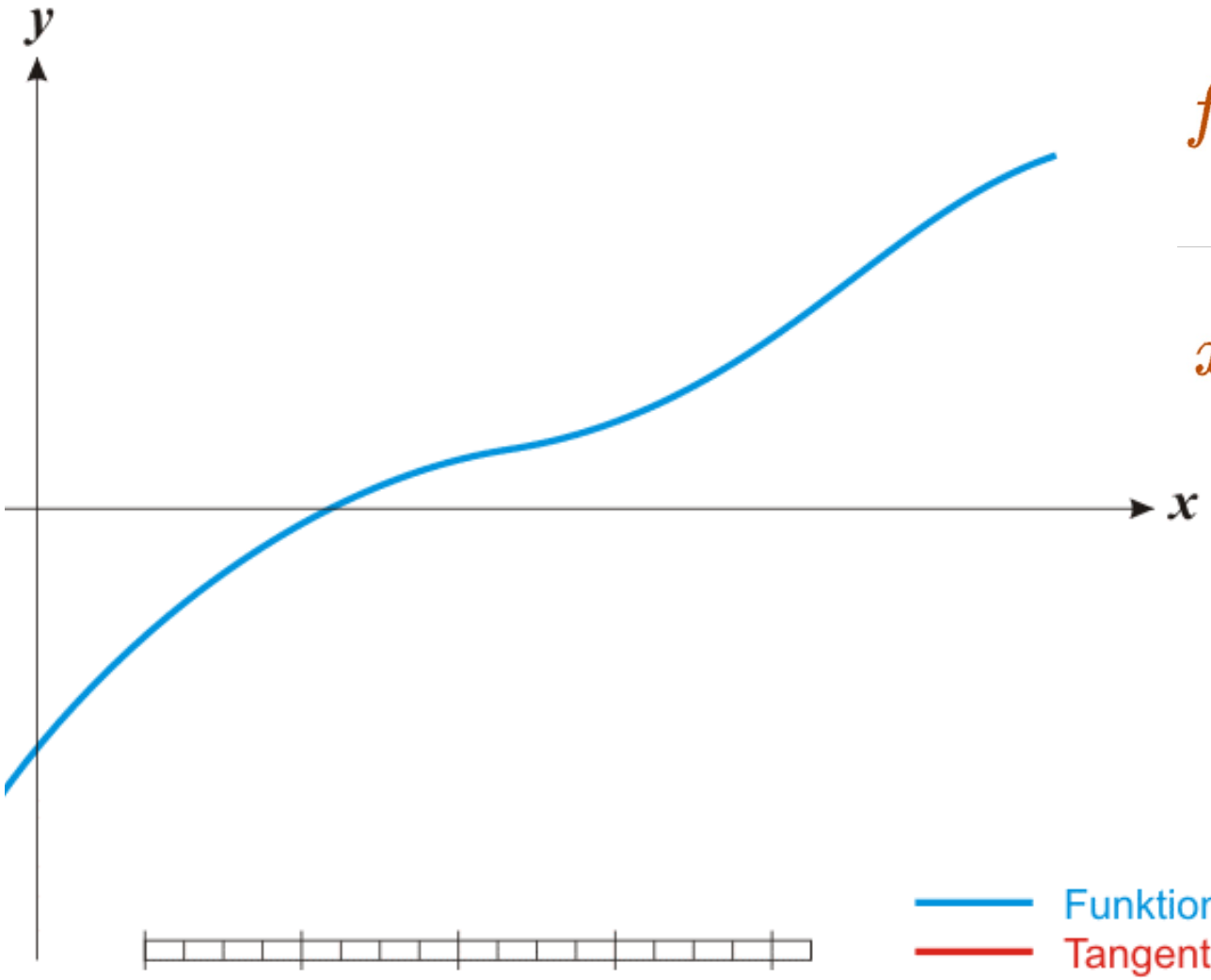
```
main_program{                    // find root of f(x) = x*x - 2.
  float xL=0,xR=2; // invariant: f(xL),f(xR) have different signs
  float xM,epsilon;
  cin >> epsilon;
  bool xL_is_positive, xM_is_positive;
  xL_is_positive = (xL*xL - 2) > 0;
  // Invariant: xL_is_positive gives the sign of f(x_L).

  while(xR-xL >= epsilon){
    xM = (xL+xR)/2;
    xM_is_positive = (xM*xM -2) > 0;
    if(xL_is_positive == xM_is_positive)
      xL = xM;  // does not { xL = xM; xL_is_positive = xM_is_positive; }
    else
      xR = xM;  // does not upset any invariant!
  }
  cout << xL << endl;
}
```



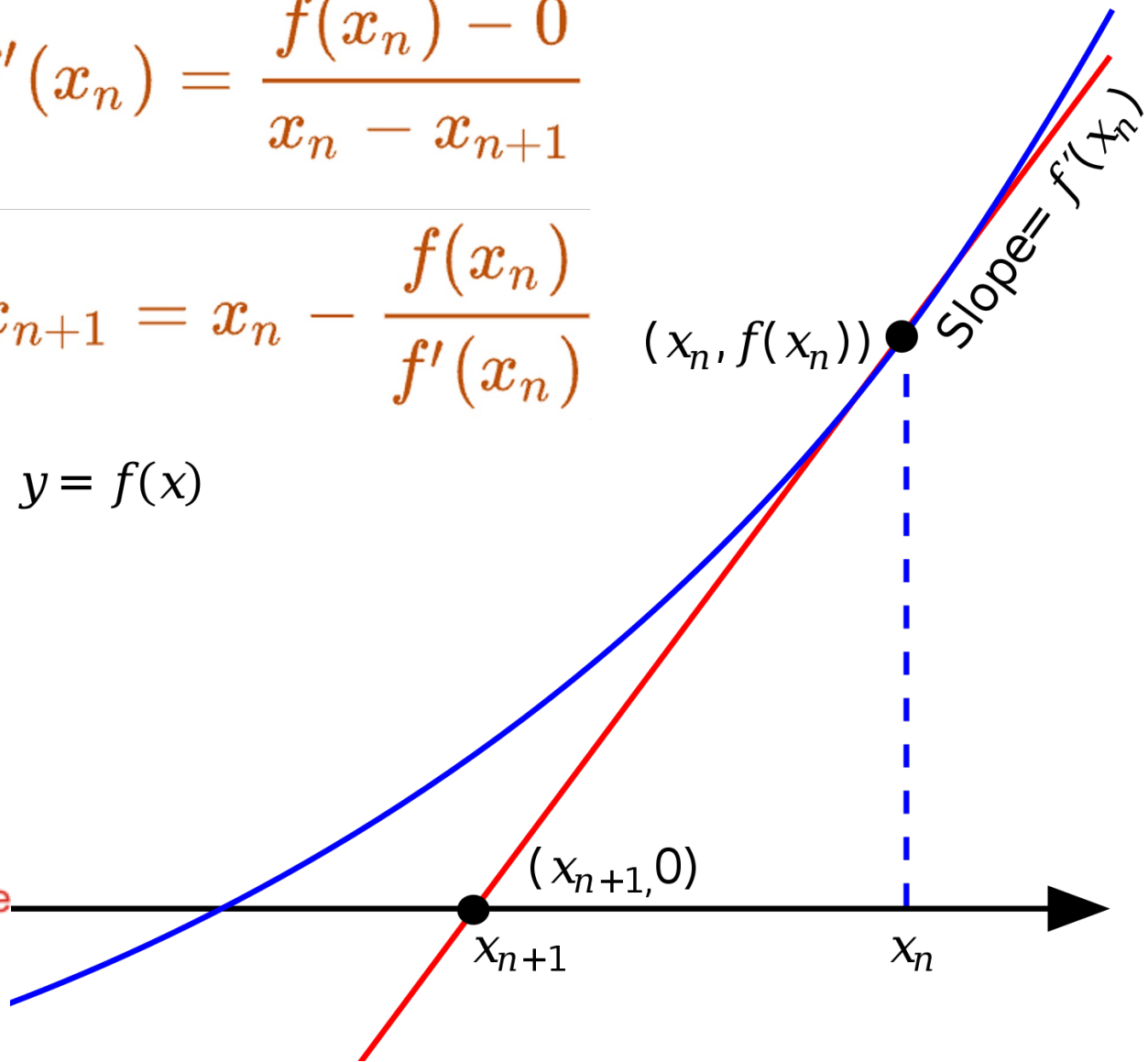
Computing Mathematical Functions

- Newton Raphson method for computing function root



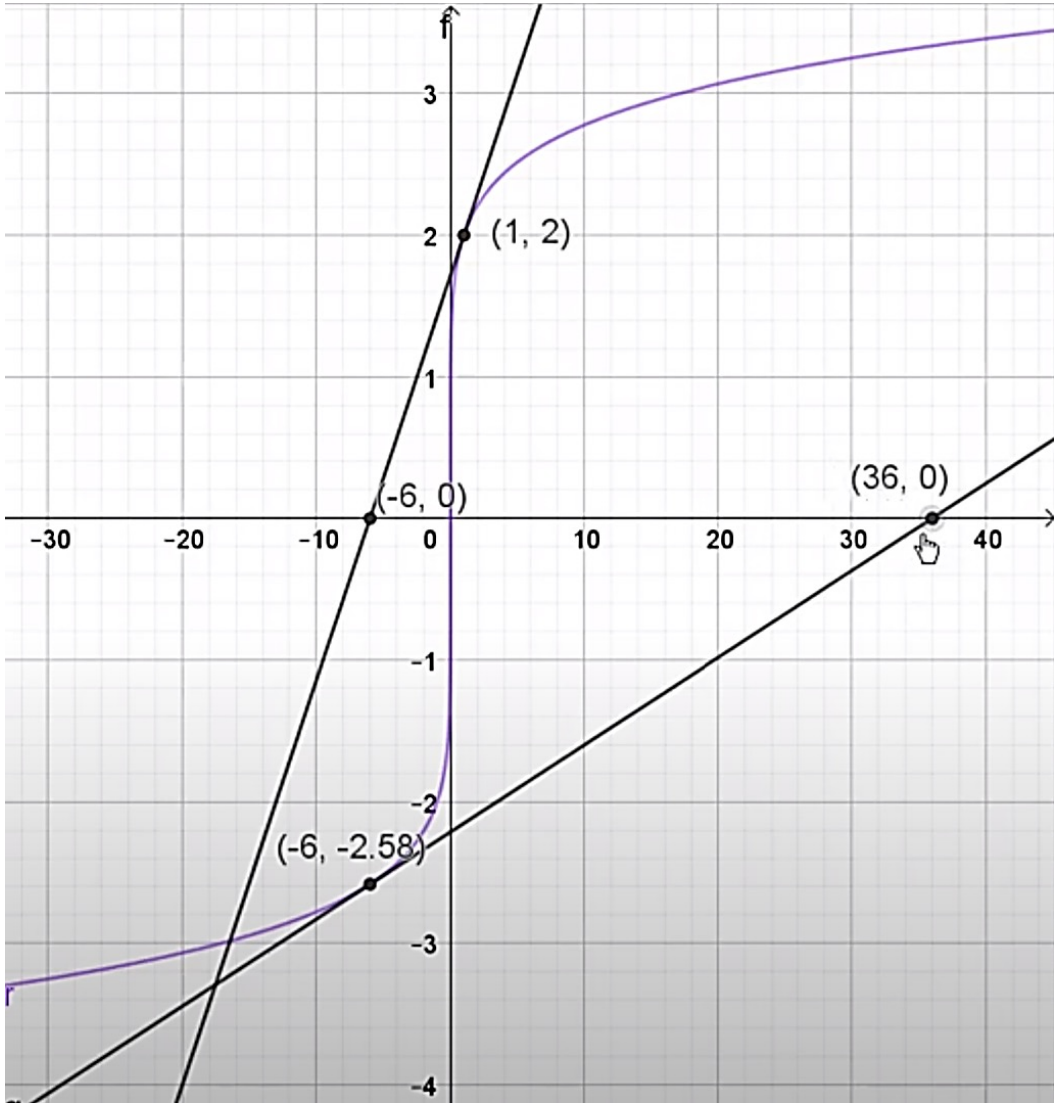
$$f'(x_n) = \frac{f(x_n) - 0}{x_n - x_{n+1}}$$

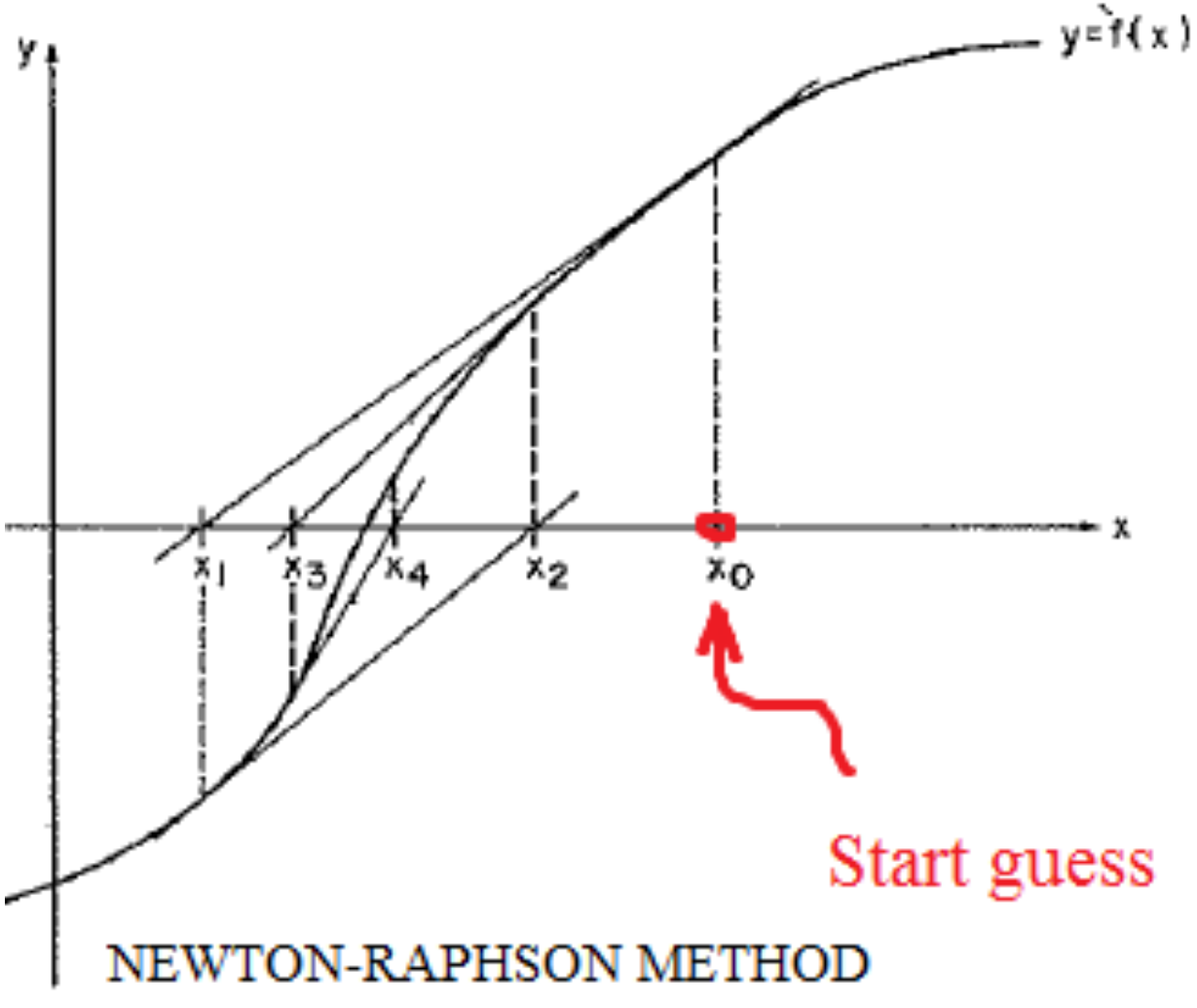
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



Computing Mathematical Functions

- Newton Raphson method for computing function root
  - Can fail to converge (depends on function and initialization), e.g., $f(x) = 2x^{(1/7)}$





Computing Mathematical Functions

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Newton Raphson method for computing function root
  - Find root of $f(x) = x^2 - y$, where y is some given (positive) constant
  - Update step: $x_{i+1} = x_i - \frac{x_i^2 - y}{2x_i} = \frac{1}{2}(x_i + \frac{y}{x_i})$

```
main_program{
  double xi=1, y;  cin >> y;
  repeat(10){
    xi = (xi + y/xi)/2;
  }
  cout << xi << endl;
}
```

You should have good initial guess.

```
main_program{
  float y; cin >> y;
  float xi=1;
  while(abs(xi*xi - y) >0.001){
    xi = (xi + y/xi)/2;
    cout << xi << endl;
  }
}
```

Computing Mathematical Functions

- Secant method
  - Start with a pair of points x0 and x1
  - Equation of “secant” line is:

$$y = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) + f(x_0)$$

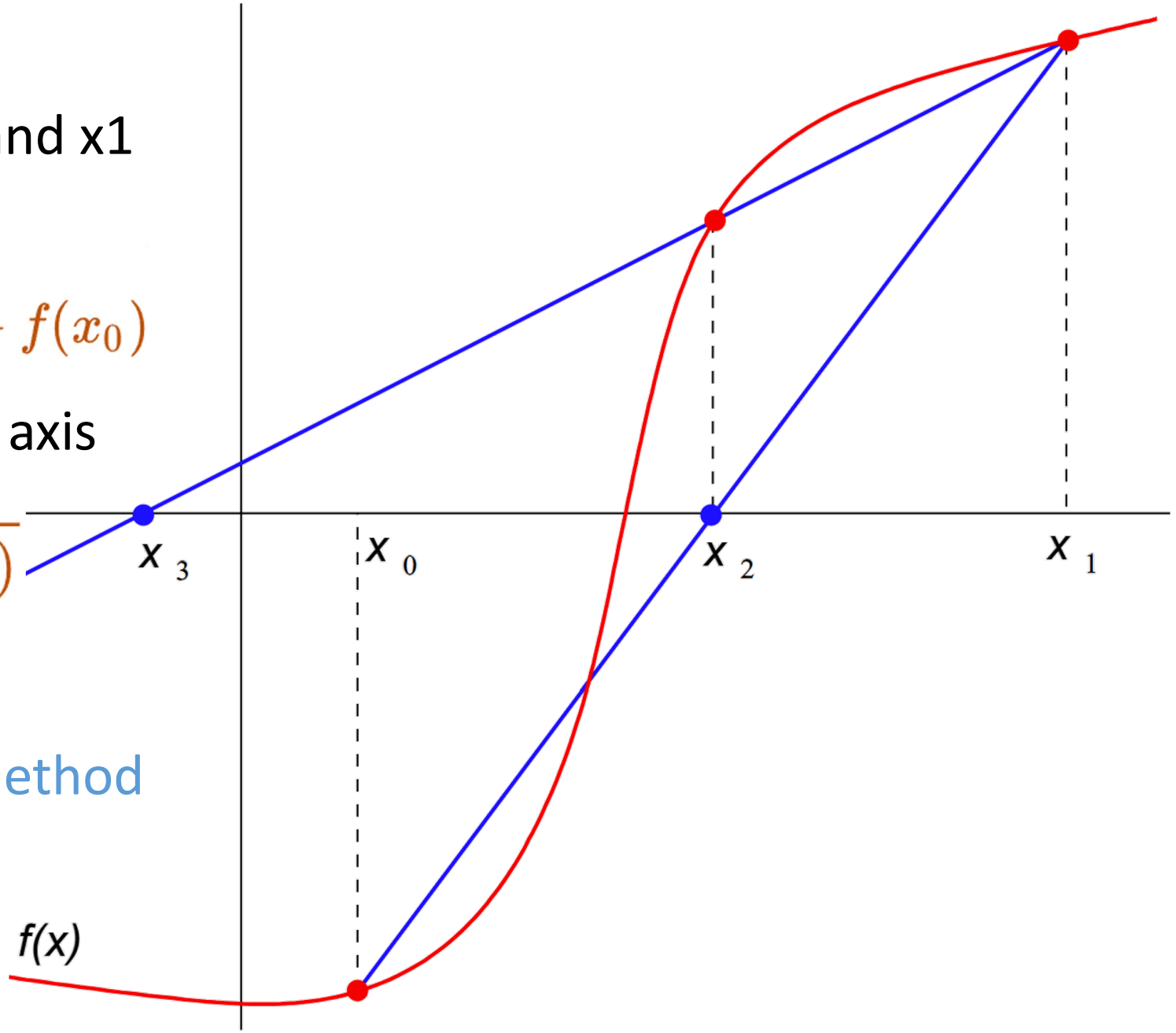
  - Find intersection of line withx axis

$$x = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

    - Call that point x2
  - Restart with pair x1 and x2
  - Similar to Newton-Raphson method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

  - Can fail to converge



Practice Examples for Lab: Set 7

- 1 Done

A more accurate estimate of the area under the curve is to use trapeziums rather than rectangles. Thus the area under a curve $f(u)$ in the interval $[p, q]$ will be approximated by the area of the trapezium with corners $(p, 0)$, $(p, f(p))$, $(q, f(q))$, $(q, 0)$. This area is simply $(f(p) + f(q))(q - p)/2$. Use this to compute the natural logarithm.

- 2 Done

Write a program to find $\arcsin(x)$ given x.

- 3 Done

Write a program that takes as input a natural number x and returns the smallest palindrome larger than x.

- 4 Done

Add checks to the GCD code to ensure that the numbers typed in by the user are positive. For each input value you should prompt the user until she gives a positive value.

Practice Examples for Lab: Set 7

- 5 Done

Simpson's rule gives the following approximation of the area under the curve of a function f:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

Use this rule for each strip to get another way to find the natural log.

- 6 Done

Children often play a guessing game as follows. One child, Kashinath, picks a number between 1 and 1000 which he does not disclose to another child, Ashalata. Ashalata asks questions of the form "Is you number between x and y?" where she can pick x, y as she wants. Ashalata's goal is to ask as few questions as possible and determine the number that Kashinath picked. Show that Ashalata can guess the number correctly using at most 10 questions. Hint: Use ideas from the bisection method.

CS 101
Computer Programming and Utilization

Functions

Suyash P. Awate

Functions

- "Function", generalizes notion of "command"
  - Some languages call it a "procedure"
- If we had a gcd() function, we could write:
- How would we create this function ?

```
main_program{
    int a=36,b=24,c=99,d=47;
    cout << gcd(a,b) << endl;
    cout << gcd(c,d) << endl;
}
```

```
int gcd                 // return-type function-name
  (int m, int n)        // parameter list: (parameter-type parameter-name ...)
{                       // beginning of function body
```

  - Function syntax
    - Function name
    - Inputs to the function: "parameters" or "arguments"
      - One or more. They have a name. They have a data type.
    - Output of the function: "return type"
    - Data processing algorithm embodied by the function: "body"
      - Parameter variables will be used within function's body
      - Body should return variable of type "return type"

Functions

- "Function", generalizes notion of "command"
  - Some languages call it a "procedure"
- If we had a gcd() function, we could write:
- How would we create this function ?

```
main_program{
   int a=36,b=24,c=99,d=47;
   cout << gcd(a,b) << endl;
   cout << gcd(c,d) << endl;
}
```

```
int gcd                 // return-type function-name
  (int m, int n)        // parameter list: (parameter-type parameter-name ...)
{                       // beginning of function body
  while(m % n != 0){
    int Remainder = m % n;
    m = n;
    n = Remainder;
  }
  return n;
}                       // end of function body
```

```
type-of-return-value function-name (parameter1-type parameter1-name,
  parameter2-type parameter2-name, ...){
  body
}
```

Functions

- What happens when control comes to "gcd(a,b)" ?
  - Get values of variables a,b
    - If arguments are expressions, evaluate them to get their values
  - Execution of main program is suspended; to be resumed later
  - Function will need to have its own area of memory
    - e.g., the first variables created in this memory are the function's parameters
    - This memory is called the activation frame, or call stack, of function call

Own Memory.....

  - Copy argument/parameter values from main program's memory region to function's memory region ("call by value", "pass by value")
  - Execute function's body
    - Commands in body can refer to only those variables that are within function's call stack
    - e.g., variable "Remainder" is "local" to function call
  - When a return statement is encountered, return-expression evaluated and value copied into activation frame of main program
  - Activation call of function is destroyed
  - Control returns to main program; execution starts from next statement

Functions By Value.....!

```
main_program{
  int a=36,b=24,c=99,d=47;
  cout << gcd(a,b) << endl;
  cout << gcd(c,d) << endl;
}
```

- Function execution: “call by value”
  - Only values of arguments passed from calling program’s frame to called function’s frame
  - Variables within main program aren’t accessible (out of scope) to code within called function; and vice versa

```
int gcd                 // retu
  (int m, int n)        // para
{                       // begi
  while(m % n != 0){
    int Remainder = m % n;
    m = n;
    n = Remainder;
  }
  return n;
}                       // end o
```

Activation frame of main_program	Activation frame of gcd(a,b)
a : 36 b : 24 c : 99 d : 47	m : 36 n : 24

(a) After copying arguments.

Activation frame of main_program	Activation frame of gcd(a,b)
a : 36 b : 24 c : 99 d : 47	m : 24 n : 12 Remainder : 12

(a) At the end of the first iteration of the loop in gcd.

Functions

- Nested function calls

```
main_program{
   cout << lcm(36,24) << endl;
}   int lcm(int m, int n){
      return m*n/gcd(m,n);
    }
        int gcd              // retu
          (int m, int n)     // para
        {                    // begi
          while(m % n != 0){
            int Remainder = m % n;
            m = n;
            n = Remainder;
          }
          return n;
        }                    // end o
```

nest *verb* (FIT INSIDE)

[I or T]

to fit one object inside another, or to fit inside in this way:

- *nested coffee tables*
- *dolls that nest inside one another*

Functions

- Specification: GCD example
  - "GCD(m,n) returns the greatest common divisor of positive integers m and n"
  - Responsibility of calling program/function
    - Must supply positive integers (pre-condition of function)
      - Can code-up a check on arguments being passed, just before calling
  - Responsibility of called program/function
    - Must return a positive integer (post-condition of function)
      - Can code-up a check on return expression
  - On the safe side:
    - Called function can also check for validity of arguments, before starting processing
    - Calling program/function can also check for validity of return value, before continuing

Functions

Specification likhna jaruri hai

- Specification: GCD example
  - Write down specification, within comments, when defining function
  - Specification is more about "what" function does, and less about "how"

```
int gcd(int L, int S)
// Function for computing the greatest common divisor of integers L, S.
// PRE-CONDITION: L, S > 0
{
...
}
```

Write down to tell other programmers what the function do

  - How the function computes the GCD is also important; comment in body
  - Why the code works is also important; also comment in body

```
// Note the theorem: If n divides m, then GCD(m,n) = n.
//                   If n does not divide m, then GCD(m,n) = GCD(n, m mod n)
```

Functions

Void

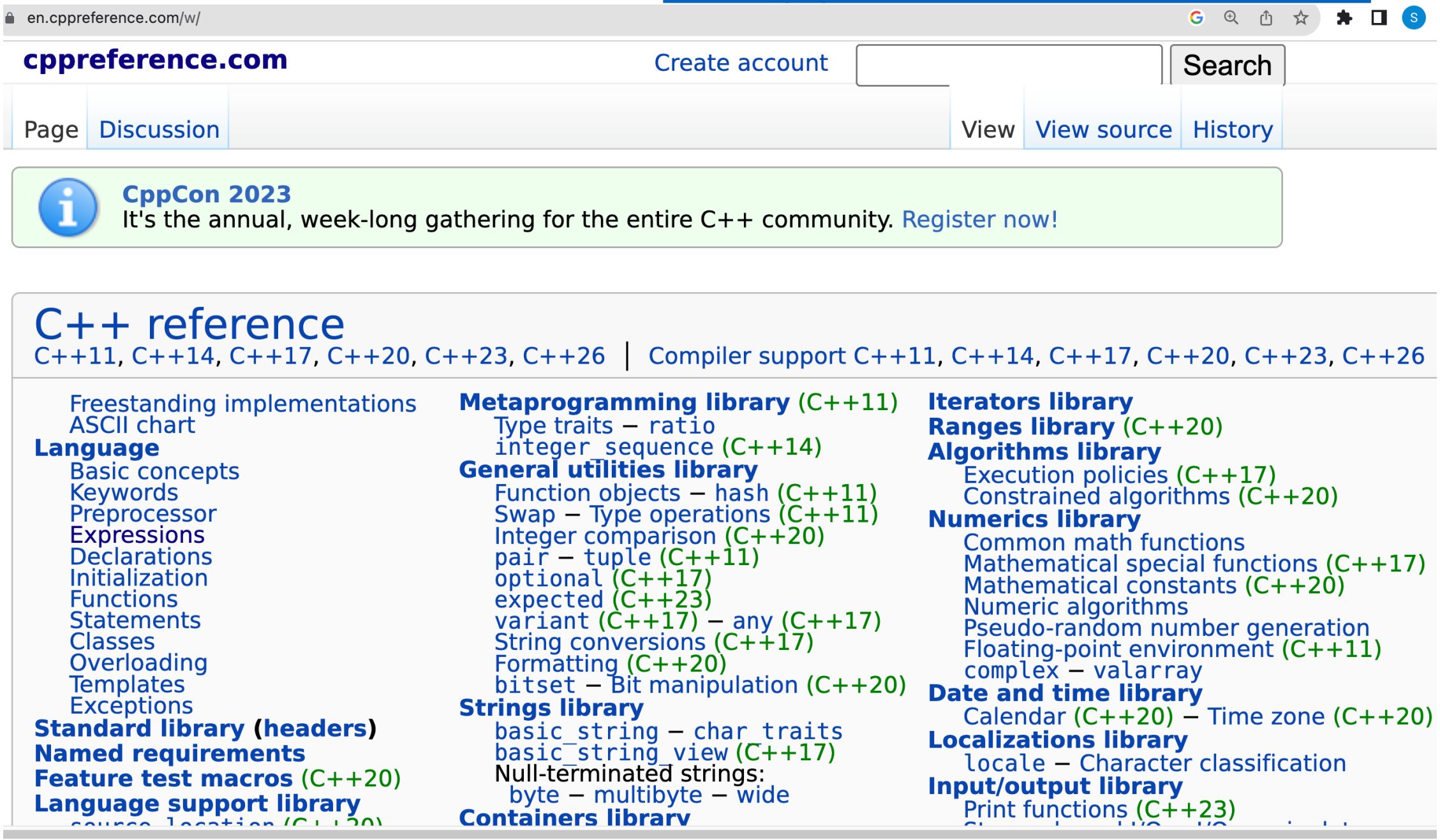
- Functions needn't return a value: return type of "void"
  - e.g., forward() function doesn't return a value
  - e.g., a function to draw a n-sided regular polygon, returning nothing

```
void polygon(int nsides, double sidelength)
// draws polygon with specified sides and specified sidelength.
// PRE-CONDITION: The pen must be down, and the turtle must be
// positioned at a vertex of the polygon, pointing in the clockwise
// direction along an edge.
// POST-CONDITION: At the end the turtle is in the same position and
// orientation as at the start.  The pen is down.
{
  for(int i=0; i<nsides; i++){
    forward(sidelength);
    right(360.0/nsides);
  }
  return;
}
```

Functions

- Main program is a function !
  - SimpleCPP camouflaged it
  - Actually when you write “main_program”,
    SimpleCPP internally replaces it by “int main ()”,
    and then passes that to C++ compiler
  - Operating system (OS) expects a function called “main” in compiled program
  - When you run program (a.out),
    OS first transfers control to main() function
  - When program ends, main() returns an integer code to OS as feedback
    - Code typically conveys if program ran properly or not (e.g., what type of error, etc.)
  - C++ allows main() without a return statement
    - 0 returned by default

C++ Resources on WWW: en.cppreference.com/w/



Functions

- Some challenges (for now)
  - Function cannot return more than 1 value
  - Suppose we write a function to swap values of 2 variables as follows:

```
void swap(int a, int b){  // will it work?
   int temp;
   temp = a;
   a = b;
   b = temp;
}
```

  - Suppose we intend to write a function read_marks_into() to update nextmark AND return bool indicating stopping criterion

```
int main(){
   double nextmark, sum=0;
   int count=0;

   while(read_marks_into(nextmark)){  // will this work?
      sum = sum + nextmark;
      count = count + 1;
   }

   cout << ‘‘The average is: ‘‘ << sum/count << endl;
}
```

Functions

Changes the value inside the main function.

- Function execution: "call by reference", "pass by reference"
  - When you want changes to a function parameter to be reflected in the corresponding variable in the calling program/function
  - Use ampersand symbol (&) prefixed to variable name in function definition

```
void Cartesian_To_Polar(double x, double y, double &r, double &theta){
  r = sqrt(x*x + y*y);
  theta = atan2(y,x);
}
```

```
int main(){
  double x1=1.0, y1=1.0, r1, theta1;
  Cartesian_To_Polar(x1,y1,r1,theta1);
  cout << r1 << ' ' << theta1 << endl;
}
```

    - What happens to values of variables x1, y1, r1, theta1 in main program before, during, after function call ?
  - Within function's activation frame, for variables passed by reference, new memory-storage area isn't allocated, but their memory region in calling program/function is used
    - No need for copying such variable's values from calling to called function

Functions

- Function execution: “call by reference”
  - Example for swapping numbers

```
void swap2(int &a, int &b){
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

```
int main{
  int x=5, y=6;
  swap2(x,y);
  cout << x << " " << y << endl;
}
```

  - Example for reading marks

```
bool read_marks_into(int &var){
  cin >> var;
  return var != 200;
}
```

```
int main(){
  double nextmark, sum=0;
  int count=0;

  while(read_marks_into(nextmark)){  // will this work?
    sum = sum + nextmark;
    count = count + 1;
  }

  cout << ‘‘The average is: ‘‘ << sum/count << endl;
}
```

Functions

&before assigning var is renaming of var.

- "Reference variables"
  - Variables passed by reference are, essentially, renamed
  - "Call by reference" also called as "call by name"
  - We can define aliases even outside context of function parameters
  - Example
    - "r" becomes a new name for "x"
    - They share memory and values
    - Changes to any one imply changes to other
    - "int &" means that variable name is going to be an alias of another variable, and that association/renaming must be defined during the declaration of the alias itself

```
int x = 10;
int &r = x;
cout << r << endl;
r = 20;
cout << x << endl;
```

  - What is the output of the code ?
    - 10
      20

Functions

- Call/pass by reference/name
- Call/pass by value

*pass by **reference***

cup =

fillCup( )

*pass by **value***

cup =

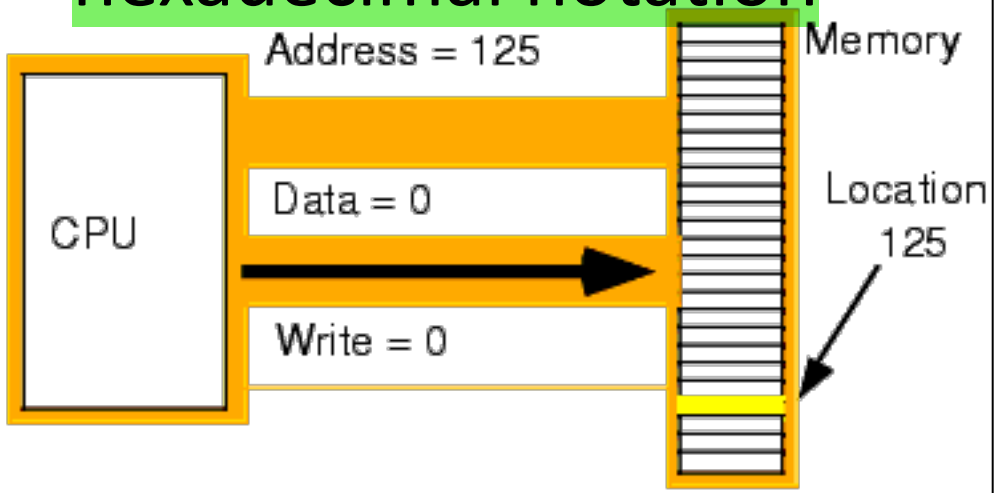
fillCup( )

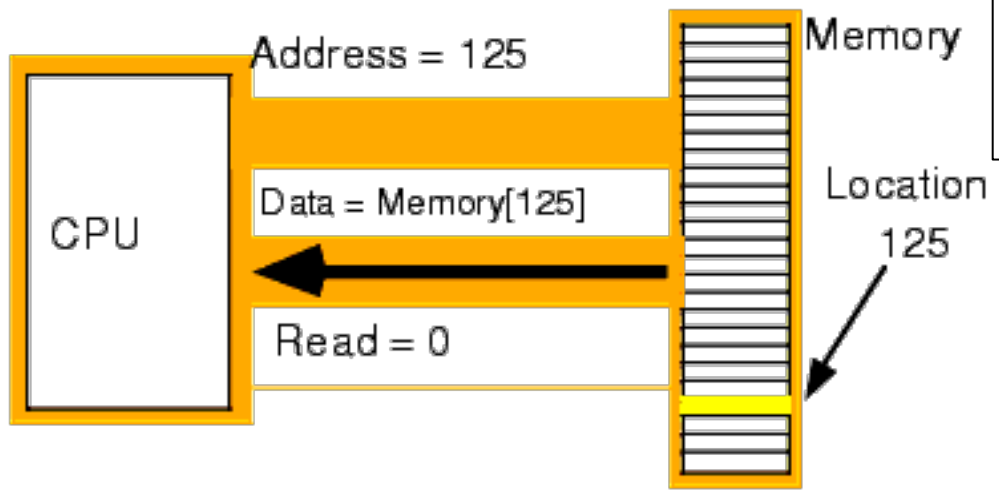
Functions

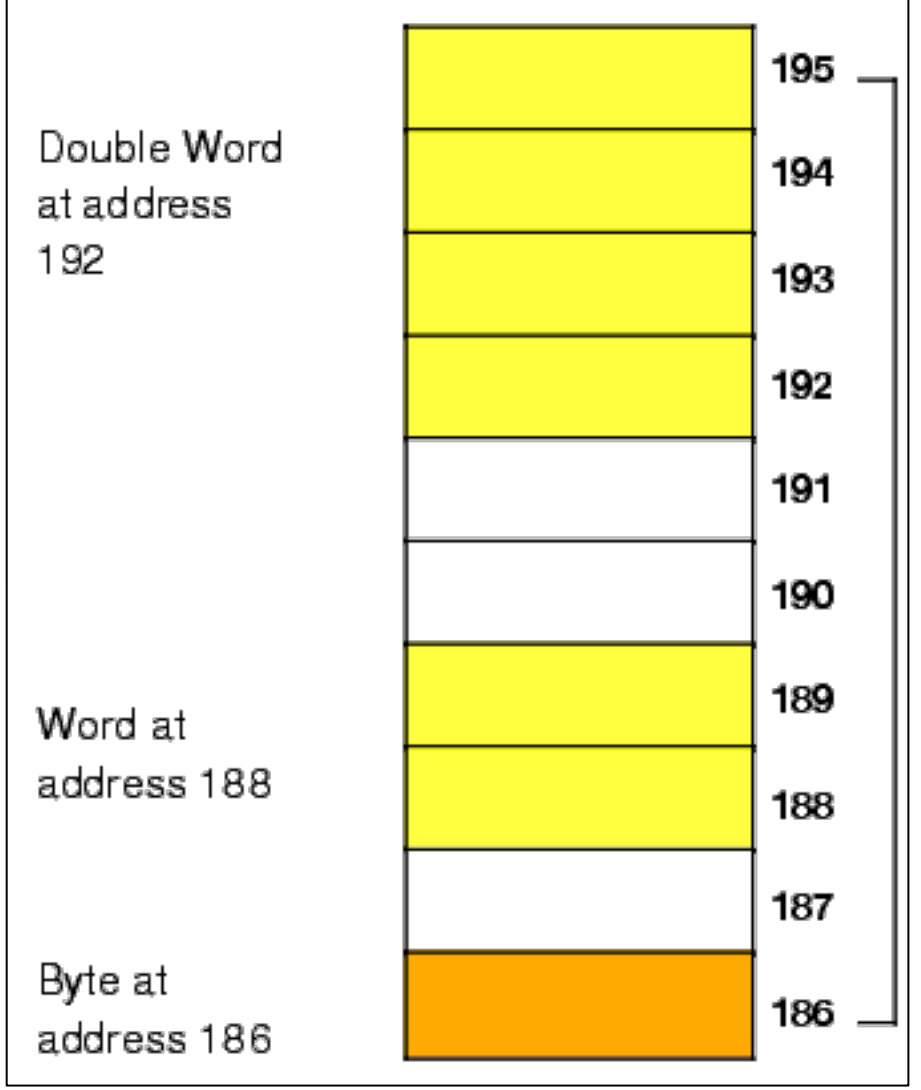
- Memory is organized into bytes (smallest unit in typical hardware)
  - Each byte has a location/address
  - Each byte may also store some data; its associated value
- When memory is allocated to a variable, it is as contiguous bytes
  - e.g., bool and char variables get 1 byte, float variable gets 4 consecutive bytes
- "Address of a variable" = address of memory location of its first byte
- C++ provides an operator to get address of a variable: ampersand (&)
  - Don't confuse this operator & from the & used to create a reference variable (e.g., "int &")
  - Unary operator; signifies "address of"
- Example:
  - "int i; cout << &i;" prints the address associated with variable i

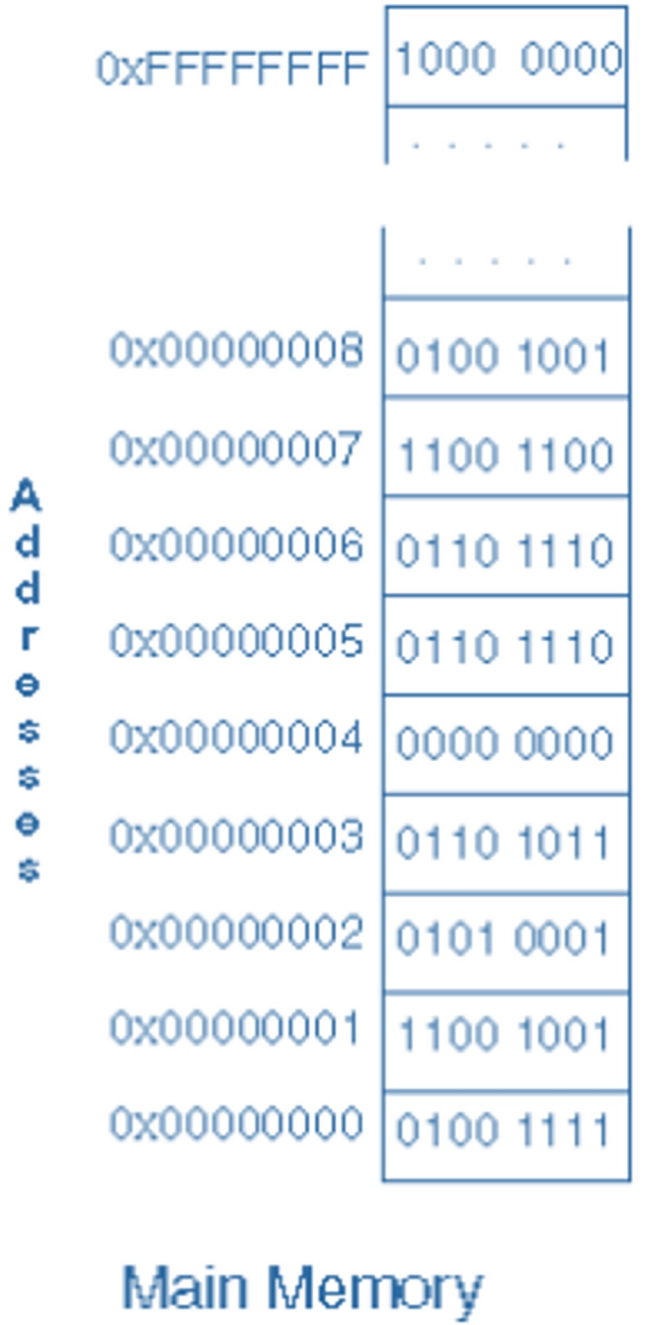
Functions

- Memory addresses often given in hexadecimal notation



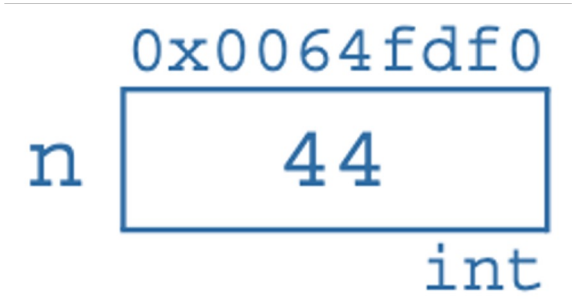






Functions

- Visualizing the representation of a variable, e.g., "int n = 44;"
  - Variable name
  - Variable type
  - Variable address
  - Variable value




```
int main()
{ int n=44;
  cout << "n = " << n << endl;     // prints the value of n
  cout << "&n = " << &n << endl;   // prints the address of n
}
```

```
n = 44
&n = 0x0064fdf0
```

Functions

- Using references

This declares rn as a reference to n:

```
int main()
{  int n=44;
   int& rn=n;  // r is a synonym for n
   cout << "n = " << n << ", rn = " << rn << endl;
   --n;
   cout << "n = " << n << ", rn = " << rn << endl;
   rn *= 2;
   cout << "n = " << n << ", rn = " << rn << endl;
}
```

```
n = 44, rn = 44
n = 43, rn = 43
n = 86, rn = 86
```

Sab kuch dono ka same hai yrr.

Like constants, references must be initialized when they are declared. But unlike a constant, a reference must be initialized to a variable, not a literal:

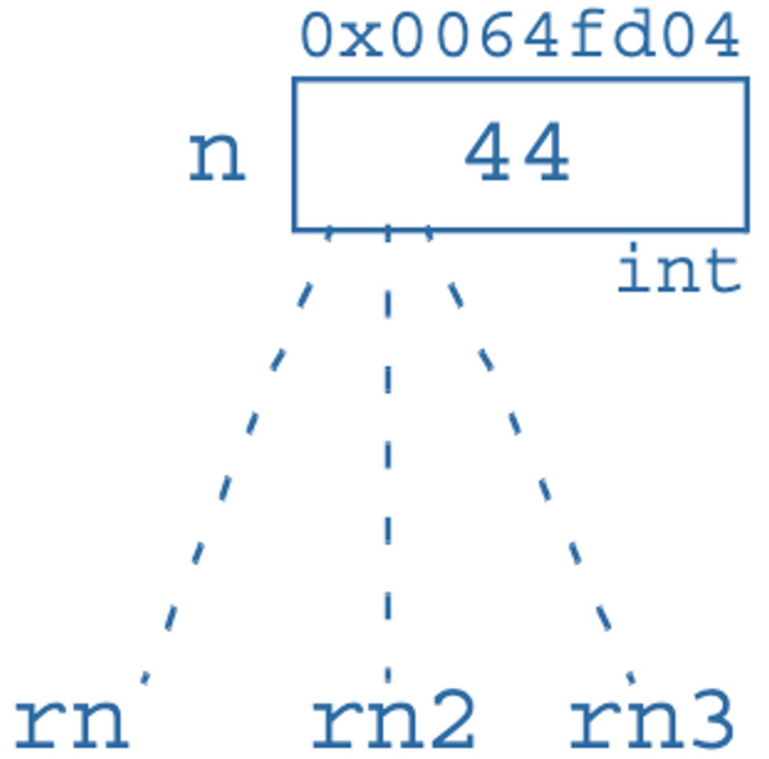
```
int& rn=44;  // ERROR: 44 is not a variable!
```

IMP to keep in mind that synonym is for variable only

Functions

Sab solmal hai bhai sab golmal hai.

- References aren't separate variables
  - "A reference to a a reference is the same as a reference to the object to which it refers" ☺
    [From Schaum's book on Programming in C++]




```
int main()
{  int n=44;
   int& rn=n;   //    is a synonym for n
   cout << "   &n = " << &n << ",   &rn = " << &rn << endl;
   int& rn2=n;    //    is another synonym for n
   int& rn3=rn;   //    is another synonym for n
   cout << "&rn2 = " << &rn2 << ", &rn3 = " << &rn3 << endl;
}
```

synonym don't have their own memory. They are just other name for original variable

```
   &n = 0x0064fde4,   &rn = 0x0064fde4
&rn2 = 0x0064fde4, &rn3 = 0x0064fde4
```

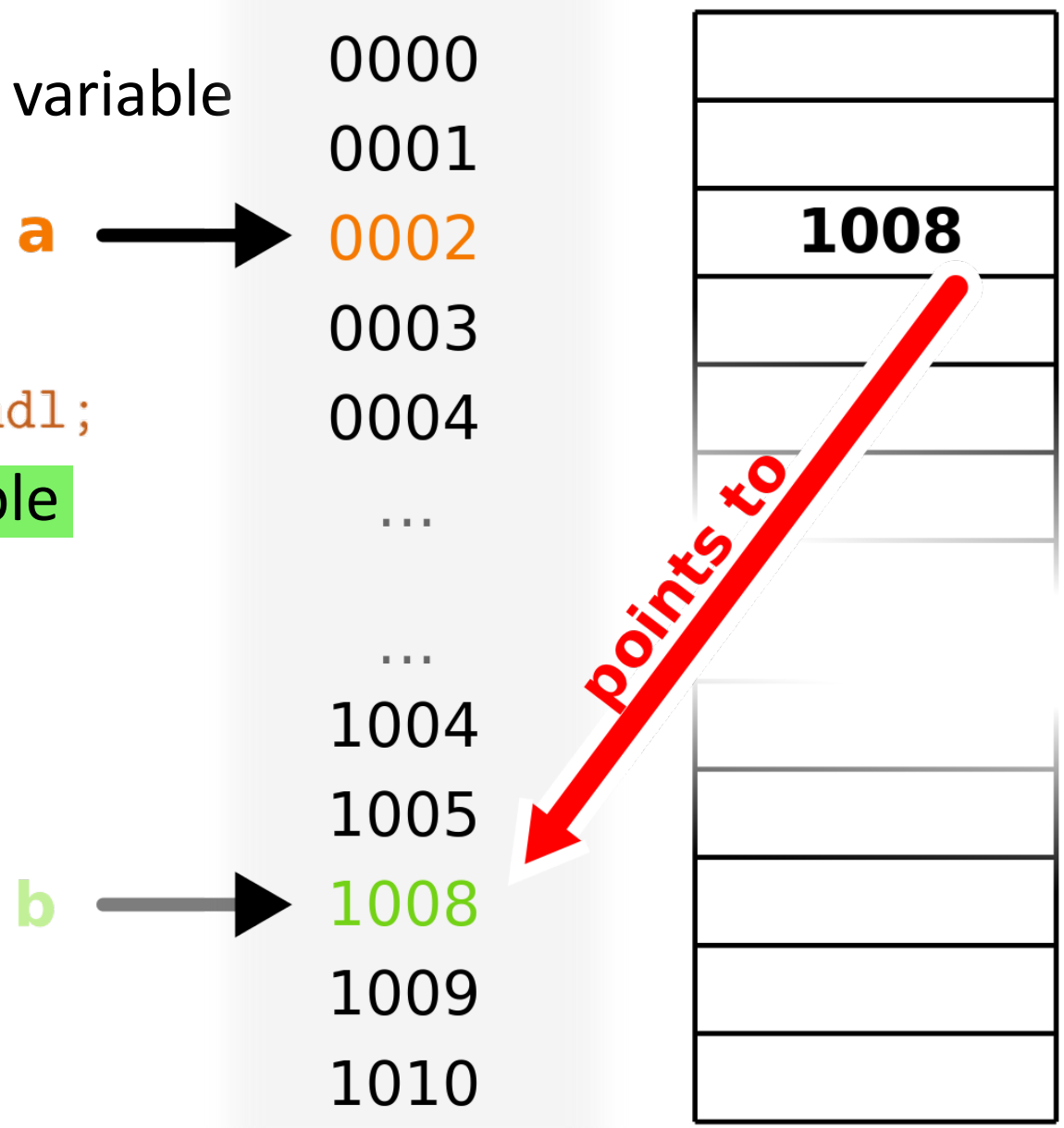
Functions

- "Pointers"
  - A variable that stores address of another variable
  - Example

```
int p=15;
int *r;
r = &p;
cout << &p << " " << r << endl;
```

  - Data type "T *" indicates a pointer variable (of derived type T *) that will store an address of variable of type T

Address	Content	Remarks
104 105 106 107	15	Allocated to p
108 109 110 111	104	Allocated to r

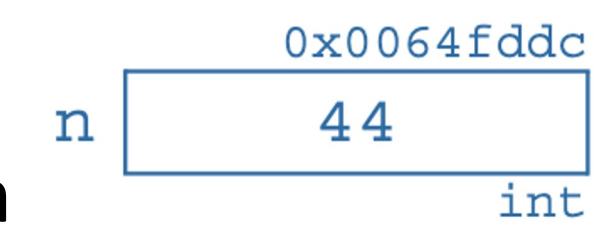


Functions

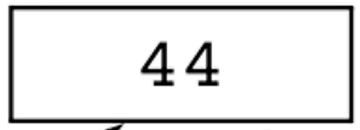
- Pointer dogs
  - Used in hunting
  - Point towards game

Functions

- Visualizing the representation of a pointer variable




```
int main()
{  int n=44;
   cout << "n = " << n << ", &n = " << &n << endl;
   int* pn=&n;  // pn holds the address of n
   cout << "       pn = " << pn << endl;
   cout << "&pn = " << &pn << endl;
}
```




```
n = 44, &n = 0x0064fddc
       pn = 0x0064fddc
&pn = 0x0064fde0
```

d, e, f, then 0

Functions

- Pointers
  - Equivalent definitions

```
int* p;     // indicates that p has type int* (pointer to int)
int * p;    // style sometimes used for clarity
int *p;     // old C style
```

  - Be careful: “int * p, q;” declares p as pointer and q as integer
- “Dereferencing” pointers
  - “reference” applied to variable gives “address” of variable, “dereference” applied to address gives back (value of) variable
  - Inverse of unary & operator is unary * operator
  - Just as operator & refers to “address of” variable, operator * refers to “variable at” address
  - Don’t confuse dereferencing * operator with “int *” declaration or multiplication operator *

Functions

- Dereferencing a pointer

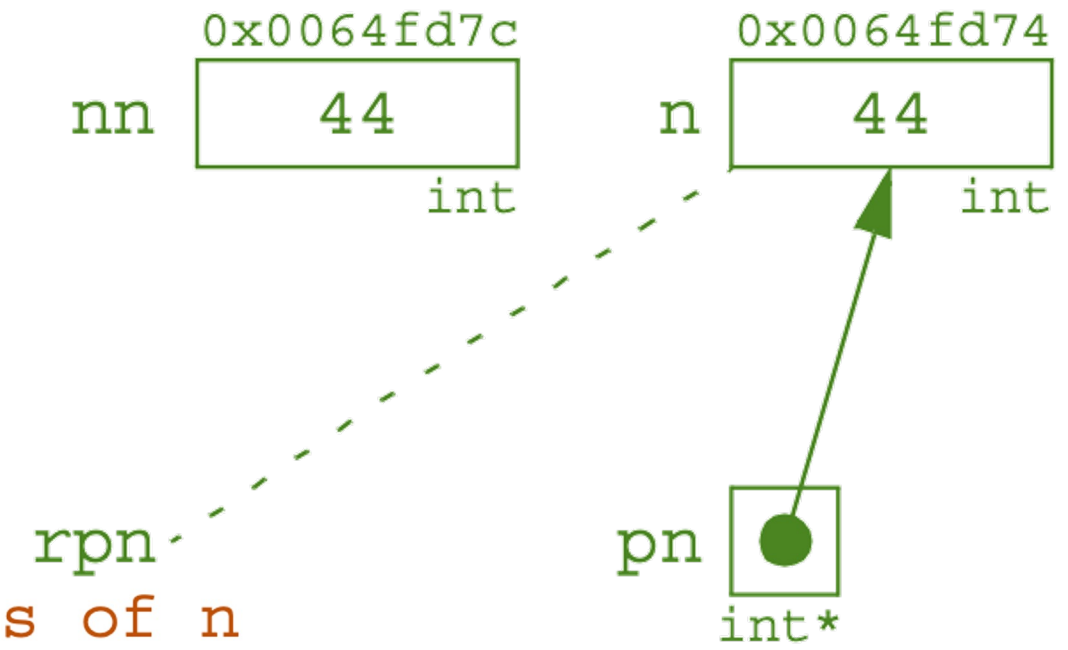
```
int main()
{  int n=44;
   cout << "n = " << n << ", &n = " << &n << endl;
   int* pn=&n;  // pn holds the address of n
   cout << "          pn = " << pn << endl;
   cout << "&pn = " << &pn << endl;
   cout << "*pn = " << *pn << endl;
}
```

```
n = 44, &n = 0x0064fdcc
        pn = 0x0064fdcc
&pn = 0x0064fdd0
*pn = 44
```

This shows that `*pn` is an alias for `n`: they both have the value 44

Functions important slide it is

- Referencing is Opposite of Dereferencing




```
int main()
{ int n=44;
  cout << "     n = " << n << endl;
  cout << "    &n = " << &n << endl;
  int* pn=&n;  // pn holds the address of n
  cout << "    pn = " << pn << endl;
  cout << "   &pn = " << &pn << endl;
  cout << "   *pn = " << *pn << endl;
  int nn=*pn;  // nn  is a duplicate of n
  cout << "    nn = " << nn << endl;
  cout << "   &nn = " << &nn << endl;
  int& rpn=*pn;  // rpn is a reference for n
  cout << "   rpn = " << rpn << endl;
  cout << " &rpn = " << &rpn << endl;
}
```

Last wala is reference

```
     n = 44
    &n = 0x0064fd74
    pn = 0x0064fd74
   &pn = 0x0064fd78
   *pn = 44
    nn = 44
   &nn = 0x0064fd7c
   rpn = 44
  &rpn = 0x0064fd74
```

Functions

- "Dereferencing" pointers
  - Example: old
  - Example: new

```
void Cartesian_To_Polar(double x, double y, double &r, double &theta){
  r = sqrt(x*x + y*y);
  theta = atan2(y,x);
}
```

```
int main(){
  double x1=1.0, y1=1.0, r1, theta1;
  Cartesian_To_Polar(x1,y1,r1,theta1);
  cout << r1 << ' ' << theta1 << endl;
}
```

Pointer is used here.

```
void CartesianToPolar(double x, double y, double* pr, double* ptheta){
  *pr = sqrt(x*x + y*y);
  *ptheta = atan2(y,x);
}
```

pointer is dereferenced here.

```
int main{
  double r,theta;
  CartesianToPolar(1.0, 1.0, &r, &theta);
  cout << r << ‘‘ ‘‘ << theta << endl;
}
```

- No parameters passed by reference; all parameters copied, i.e., value 1 into x, value 1 into y, address of r into pr, address of theta into ptheta
  - This style of calling a function called "pass by address"

Functions

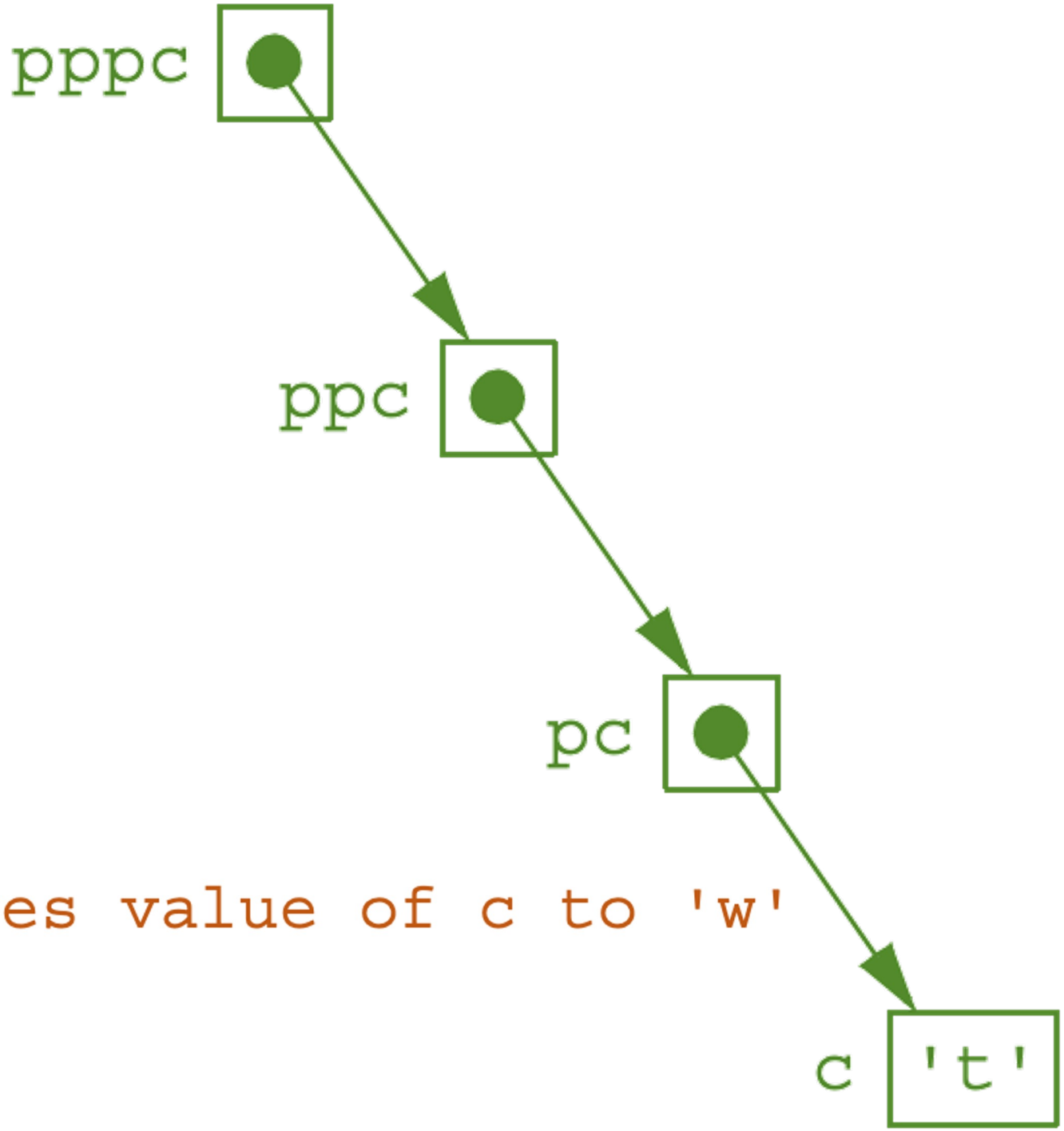
- "Dereferencing" pointers
  - Example

```
void swap(int* pa, int* pb){
   int temp;
   temp = *pa;
   *pa = *pb;
   *pb = temp;
}

int main{
   int x=5, y=6;
   swap(&x,&y);
   cout << x << ‘‘ ‘‘<< y << endl;
}
```

Functions

- Pointers to pointers
  - [From Schaum's book on Programming in C++]




```
char c = 't';
char* pc = &c;
char** ppc = &pc;
char*** pppc = &ppc;
***pppc = 'w';  // changes value of c to 'w'
```

dereferenced it.

Functions

what does increment of pointer means here.

- Constant pointer. Pointer to a constant.
  - [From Schaum's book on Programming in C++]

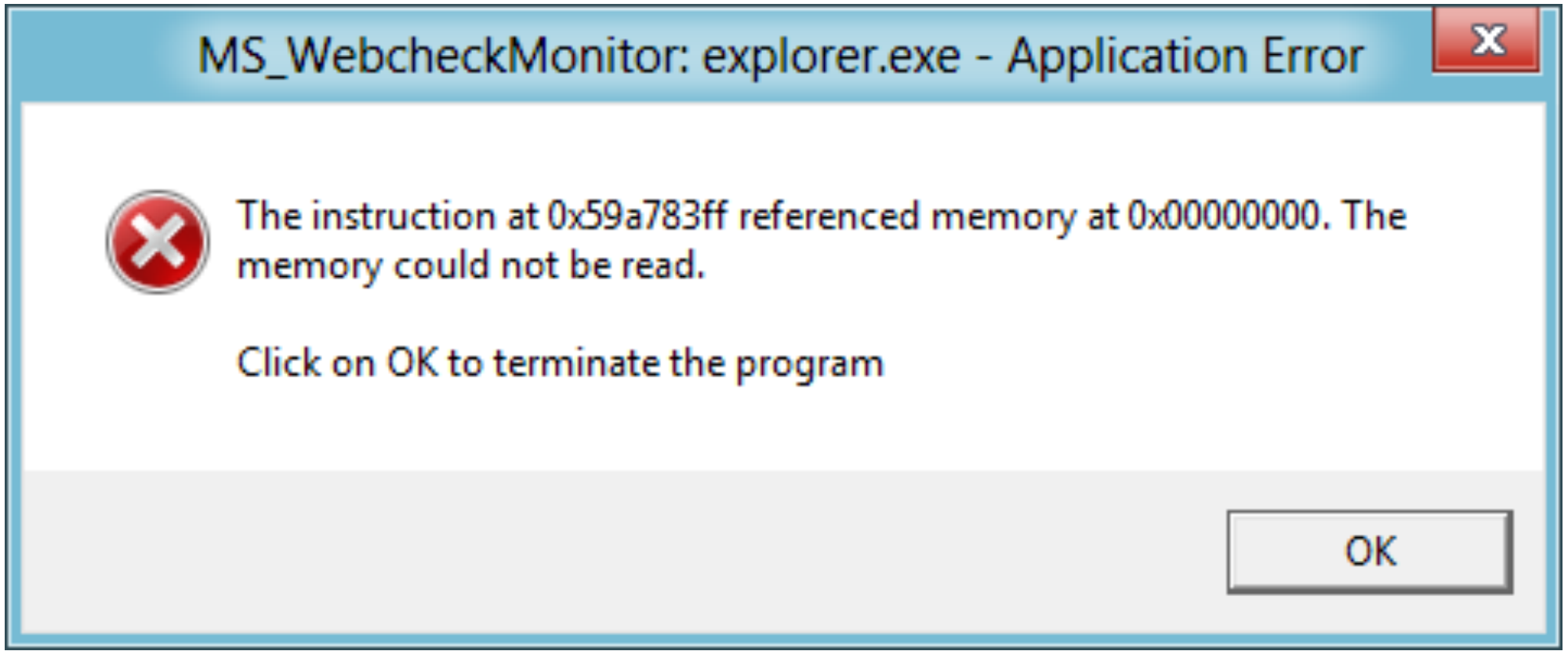
```
int n = 44;                          // an int
-------------------------------------------------------------
int* p = &n;                         // a pointer to an int
++(*p);                              // ok: increments int *p
++p;                                 // ok: increments pointer p
-------------------------------------------------------------
int* const cp = &n;                  // a const pointer to an int
++(*cp);                             // ok: increments int *cp
++cp;                                // illegal: pointer cp is const
-------------------------------------------------------------
const int k = 88;                    // a const int
const int * pc = &k;                 // a pointer to a const int
++(*pc);                             // illegal: int *pc is const
++pc;                                // ok: increments pointer pc
-------------------------------------------------------------
const int* const cpc = &k;           // a const pointer to a const int
++(*cpc);                            // illegal: int *cpc is const
++cpc;                               // illegal: pointer cpc is const
```

Functions

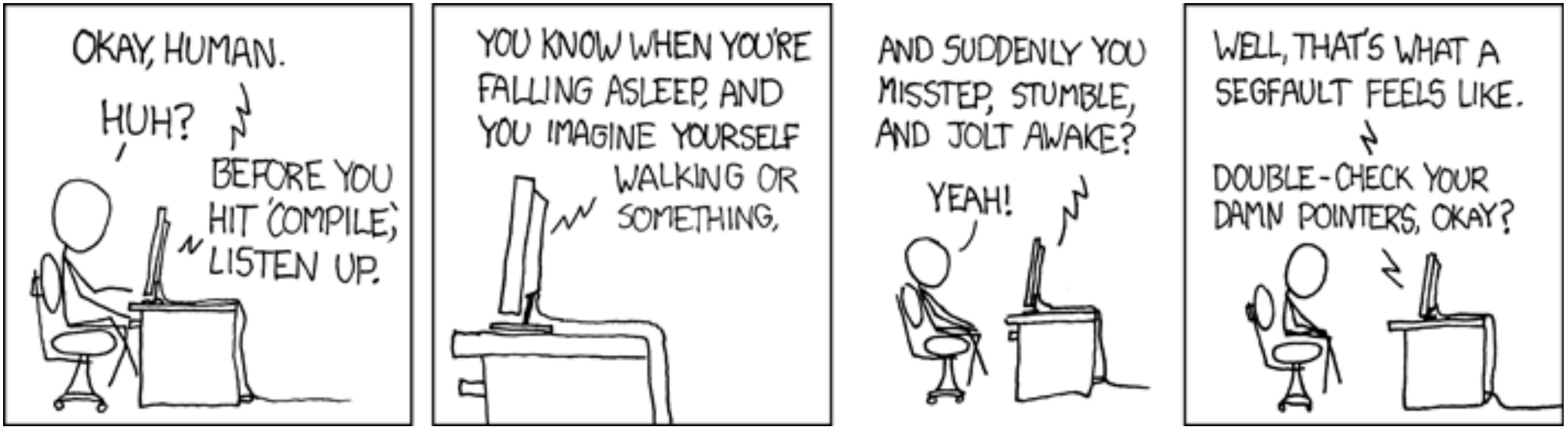
- Call-by-reference (passing aliases) versus call-by-address (passing pointers)
  - Call-by-reference implicitly (behind the scenes):
    - Passes addresses of variables to function
    - Performs dereferencing inside the function
  - Passing addresses and dereferencing pointers requires utmost care lest the addresses aren't accessible by program (e.g., address of zero= NULL) leading to a program error
    - Such an error is called a "segmentation fault"
    - en.wikipedia.org/wiki/Segmentation_fault
    - "A segmentation fault (often shortened to segfault) or access violation is a fault, or failure condition, raised by hardware with memory protection, notifying an operating system
      the software has attempted to access a restricted area of memory (a memory access violation)"

Functions

- Segfault
  - en.wikipedia.org/wiki/Segmentation_fault
  - “Common class of error in programs written in languages like C that provide low-level memory access and few to no safety checks”
  - A null pointer dereference on Windows 8



Functions


OKAY, HUMAN.
HUH?
BEFORE YOU HIT 'COMPILE', LISTEN UP.
YOU KNOW WHEN YOU'RE FALLING ASLEEP, AND YOU IMAGINE YOURSELF WALKING OR SOMETHING.
AND SUDDENLY YOU MISSTEP, STUMBLE, AND JOLT AWAKE?
YEAH!
WELL, THAT'S WHAT A SEGFAULT FEELS LIKE.
DOUBLE-CHECK YOUR DAMN POINTERS, OKAY?

Practice Examples for Lab: Set 8

- 1 done, omit the last value pass then they are passed by reference.

The k-norm of a vector (x, y, z) is defined as $\sqrt[k]{x^k + y^k + z^k}$. Note that the 2-norm is in fact the Euclidean length. Indeed, the most commonly used norm happens to be the 2 norm. Write a function to calculate the norm such that it can take k as well as the vector components as arguments. You should also allow the call to omit k, in which case the 2 norm should be returned.

- 2 Done

Write a function to find the cube root of a number using Newton's method. Accept the number of iterations as an argument.

- 3 Done

Modify the function `polygon` so that it returns the perimeter of the polygon drawn (in addition to drawing the polygon).

- 4

Write the function `read_marks_into` and the main program for mark averaging using pointers.

Practice Examples for Lab: Set 8

- 5 Done

A key rule in assignment statements is that the type of the value being assigned must match the type of the variable to which the assignment is made. Consider the following code:

```
int *x,*y, z=3      ;
```

`y = &x;` y is a pointer to an int, not a pointer to a pointer.

`z = y;` In C++, you generally can't assign an address (pointer) to an integer variable

`y = *x;` x has not been initialized and does not point to any valid memory location, dereferencing it will lead to undefined behavior, which could be a crash or some other unexpected result.

Each of the assignments is incorrect. Can you guess why? If not, write the code in a program, compile it, and the compiler will tell you!