Exploring Computer Science Concepts

Via ACSL Competitions

Number Systems

- Decimal
  - base 10
- Binary
  - base 2
- Octal
  - base 8
- HexaDecimal
  - base 16

Digits per base

- Decimal
  - Ten digits
  - 0 1 2 3 4 5 6 7 8 9
- Binary
  - Two Digits
  - 0 1
- Octal
  - 8 digits
  - 0 1 2 3 4 5 6 7
- Hexadecimal
  - 16 digits
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F

Counting in Decimal

- Increment the lowest place value (right most digit)
- When last digit is reached
  - Set the current column to 0
  - Increment the column on the left by 1
- Lets count with decimal
  - 0 to 10
  - 95 to105
- How frequently do we add a new digit ?

Decimal	Decimal
0	95
1	96
2	97
3	98
4	99
5	**100**
6	101
7	102
8	103
9	104
10	105

Counting in other bases

- Counting in binary
  - We add a new digit frequently
    - At 2 , 4, 8, 16 … decimal values
- Counting in Octal
  - We add a new digit at every
    - At 8, 64,128 …
- Counting in HexaDecimal
  - We add a new digit at every
    - At 16, 256 … values

Decimal	Binary	Octal	HexaDecimal
0	0	0	1
1	1	1	1
2	**10**	2	2
3	11	3	3
4	**100**	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	**1000**	**10**	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	**10000**	**20**	**10**

Place Values

- Decimal
  - *We deal in power of 10s*
  - 2452 = 2 * 1000 + 4 * 100 + 5 * 10 + 2 * 1
  - 101 = 1 * 100 + 0 * 10 + 1 * 1
- Binary
  - *We deal in power of 2s*
  - 111 = 1 * 4 + 1 * 2 + 1 * 1
  - 1010 = 1 * 8 + 0 * 4 + 1 * 2 + 0 * 1

Place values
Converting binary →decimal

Decimal	Binary
2452 ➔ 2 x 10^3 = 2000 4 x 10^2 = 400 5 x 10^1 = 50 2 x 10^0 = 2 = 2452	111 ➔ 1 x 2^2 = 1 x 4 = 4 1 x 2^1 = 1 x 2 = 2 1 x 2^0 = 1 x 1 = 1 = 7
101 ➔ 1 x 10^2 = 100 0 x 10^1 = 0 1 x 10^0 = 1 = 101	1010 ➔ 1 x 2^ 3 = 1 x 8 = 8 0 x 2^ 2 = 0 x 4 = 0 1 x 2^ 1 = 0 x 2 = 2 0 x 2^ 0 = 0 x 1 = 0 = 10
756 ➔	1000 ➔

Converting Binary to Decimal

128	64	32	16	8	4	2	1	
	1	0	0	1	0	0	1	= 64 + 16 + 1 = 81
1	0	0	0	0	1	1	1	= 128 + 4 + 2 + 1 = 135
1	1	1	0	0	0	0	0	= ?

Converting Decimal to Binary

128	64	32	16	8	4	2	1

= 22

= 33

= 130

Exercises

- Convert following Decimal numbers to binary
  - 127_{10}
  - 128_{10}
  - 129_{10}
  - 255_{10}
  - 256_{10}
- Convert following Binary numbers to Decimal
  - 101101_2
  - 1110_2
  - 1111_2
  - 0110_2

Challenge

- How does binary addition and subtraction work ?

Place values
Converting octal,hexadecimal→decimal

Octal → Decimal	HexaDecimal → Decimal
777 ➔ 7 x 8^2 = 7 x 64 = 448 7 x 8^1 = 7 x 8 = 56 7 x 8^0 = 7 x 1 = 7 = 511	2AB ➔ 2 x 16^2 = 2 x 256 = 512 A x 16^1 = 10 x 16 = 160 B x 16^0 = 11 x 1 = 11 = 683
137 ➔ 1 x 8^2 = 1 x 64 = 64 3 x 8^1 = 3 x 8 = 24 7 x 8^0 = 7 x 1 = 7 = 95	101 ➔ 1 x 16^2 = 1 x 256 = 256 0 x 16^1 = 0 x 16 = 0 1 x 16^0 = 1 x 1 = 1 = 257
756 ➔	4A3 ➔

Converting Decimal to Octal

$(312)_{10}$ → $(?)_8$

312 / 8 → quotient : 39 , Remainder 0

39 / 8 → quotient : 4 , Remainder 7

4 / 8 → quotient : 0 , Remainder 4

$(470)_8$

$(112)_{10}$ → $(?)_8$

112 / 8 → quotient : 14 , Remainder 0

14 / 8 → quotient : 1 , Remainder 6

1 / 8 → quotient : 0 , Remainder 1

$(160)_8$

Exercises

- Convert following Decimal numbers to Octal
  - 111_{10}
  - 88_{10}
  - 511_{10}
  - 512_{10}
  - 513_{10}
- Convert following Octal numbers to Decimal
  - 45_8
  - 77_8
  - 100_8
  - 101_8

Converting Decimal to Hexadecimal

$(312)_{10} \rightarrow (?)_8$

312 / 16 → quotient : 19 , Remainder 8

19 / 16 → quotient : 1 , Remainder 3

1 / 16 → quotient : 0 , Remainder 1

$\mathbf{(138)_{16}}$

$(112)_{10} \rightarrow (?)_8$

112 / 16 → quotient : 7 , Remainder 0

7 / 16 → quotient : 0 , Remainder 7

$\mathbf{(70)_{16}}$

Hexadecimal → Octal

$(AC)_{16}$ → $(\mathbf{1010})_2$ x 16 + $(\mathbf{1100})_2$ x 1 //Replace hexa with binary

→ $(\mathbf{1010\ 1100})_2$ // Convert to binary

→ $(\mathbf{010\ 101\ 100})_2$ // group by 3s , added extra 0s in the front

→ $(\ \mathbf{254})_8$// Replace each group by octal value

$(1EF)_{16}$ → $(\mathbf{0001})_2$ x 256 + $(\mathbf{1110})_2$ x 16 + (1111) x 1

→ $(0001\ 1110\ 1111)_2$ // Convert to binary

→ $(\cancel{000}\ 111\ 101\ 111)_2$ // group by 3 , removed 0s in the front

→ $(\ \mathbf{757})_8$// Replace each group by octal value

Octal → Hexadecimal

- (**757**)$_8$ → (?)$_{16}$
- (**254**)$_8$ → (?)$_{16}$

Exercises

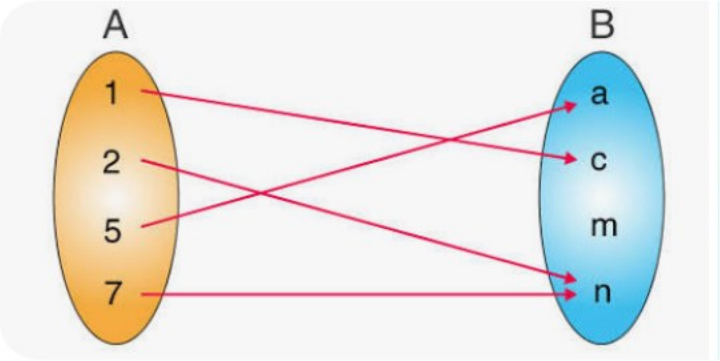
- Convert following Decimal numbers to HexaDecimal/Octal
  - 255_{10}
  - 256_{10}
  - 257_{10}
- Convert following HexaDecimal numbers to Decimal, Octal
  - 99_{16}
  - 100_{16}
  - 101_{16}

Recursion : programming

- Create factorial
  - 5! = 5 x 4 x 3 x 2 x 1 = **120**
  - 2! = **2**
- Provide sum of Fibonacci numbers using recursion
  - fibonacci(13)
    - 0 + 1 + 1 + 2 + 3 + 5 + 8 + 13 = **33**
  - fibonacci(3)
    - 0 + 1 + 1 + 2 + 3 = **7**

Relations

- A **relation** : connection/mapping between elements of two or more sets, Some characteristics …
  - It is a **mapping** as *shown in figure*
  - Can be written as **Ordered pairs**
    - {(1,c), (2,n),(5,a),(7,n)}
  - Not always unique
    - E.g. $y^2 = 4$ has multiple solutions (how many?)



- Examples
  - Numerical relationship : 4+3 = 7
  - Equation : y = 2x+3
  - Geometry : two congruent triangles
  - Set theory : A is a subset of B

Functions , mathematical kind

- A **relation** that gives exactly **one unique output for each input**

x → $\boxed{\text{Rule}}$ → y :this is a function as you always get one answer

x → $\boxed{\text{Rule}}$ → a, b, c :Not a function as you get multiple answers

- Representation
  - f(x) = 2x+1 : this is a function
  - g(x) = ± 3x : this is **not** a function , why?
- Follow up reading (optional): pg 11-13 : [functions]

Evaluating functions

- Solving
  - Substitute variables with numerals
  - Evaluate
    - Follow PEMDAS/BODMAS
- Solve for $x = 0,1,2,3$
  - $f(x) = 3x + 1$
  - $g(x) = 2x^2 + 3$
  - $h(x) = x^2 + 2x +1$

Recursive Functions

- Functions calling themselves

Fibonacci numbers	$fib\colon \mathbb{N} \to \mathbb{N}$ $fib(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ fib(n-1) + fib(n-2), & \text{if } n \geq 2. \end{cases}$
Factorial	**fact(n) = n * fact(n-1)** {given, *fact(1) = fact(0) =1}*
Lucas numbers (*same rule as Fibonacci but with different starting values*)	**l (n) = l(n-1) + l(n-2)** {given , *l(0) = 2, l(1) = 1}*

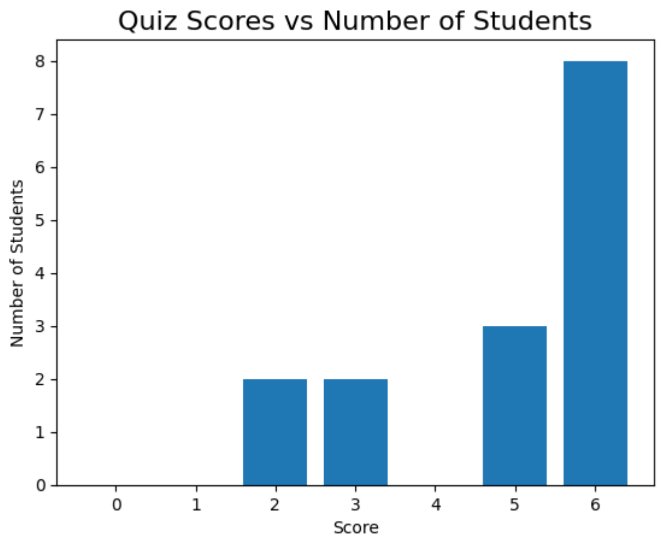
- Evaluate for ‘n’ = 2 , 4, 5, 6
- Follow up reading (optional): nested functions visualized

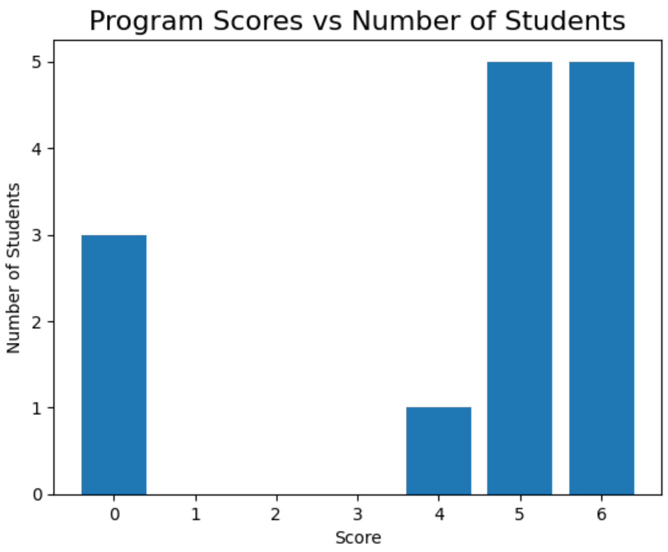
Programming, , Test #1

- Need to know
  - Data types
  - Conditionals : If/else
  - Loops : for
  - Arrays : 2 dimensional
- Practice
  - Arrays- DS (Hackerrank)
    - Do the 4 problems marked “easy”
    - Optional :“hard”, “medium” one are for extra challenge

Test 1, observations

- Great participation, nearly everyone attempted!
- Overall students did better in quiz
  - 8x full marks in quiz vs 5x full marks in programming
- 4 students got a zero in programming (with 1 not attempted)





Test 2 , analysis

- The quizzes
  - seem to be easier for students to master
  - Success cause everyone attempted, next we figure out how to raise scores there
    - Suggestions ?
- Programming
  - It was amazing attempt
  - The codes generally worked
  - But 4/15 students had zeroes.
  - Suggestions ?
- Goal :We need raise scores for everyone and move some in attempted stage
  - Note
    - Compete with yourself , Do better than your last attempt
    - Ask for help
      - on what you don’t know, to learn it.
      - And even on what you know , i.e. to master it

Topics for Test 2

- Mathematical expressions
  - Representations (infix, prefix, postfix)
  - Evaluation or expression
- Discrete Mathematics
  - Logical Operators (AND , OR , NOT, SHIFT,…)
  - Boolean logic
- Programming
  - char, string ,arrays
  - Loops
  - Conditionals
    - If/else
    - switch/case

Infix expressions

- Encounter them in grade maths. e.g.
  - (11+14)/(9 - 3) + 2
  - 3+7 / (4 * 5 - 6)
  - - 2 + 8 / 2
- Can also be written using variables
  - (a + b) / (c – d) + e
  - x + y / (g * h – k)
- Evaluated using
  - PEMDAS / BODMAS

Expression : anatomy

- Expression is made of
  - Operators (+ , - , * , /)
  - Operands (numbers, variables)
- Unary operators : - a
  - <operator> <operand>
- Binary operators (infix) : a + b
  - < operand> <operator> <operand>

Prefix/Postfix expressions

- Benefits
  - Remove ambiguity(i.e. can forget PEMDAS, BODMAS)
  - Computer friendly
  - Works with both unary/binary
  - Faster evaluation
- Expression styles
  - Infix : operator in **middle** of operands
    - (5+6)*3
    - < ộřêsắŋđ> <*operator*> <ộřêsắŋđ>
  - Prefix : operator **before** operands
    - * + 5 6 3
    - <*operator*> <ộřêsắŋđ> < ộřêsắŋđ>
  - Postfix : operator **after** operands
    - 5 6 + 3 *
    - <ộřêsắŋđ> < ộřêsắŋđ> <*operator*>

Evaluating expression,
*but first **Stacks** and **Queues***

- What is a **Stack** ?
  - This is a sequential structure of **items**
  - That are **pushed at one end**
  - **Pulled** via the **same end**
  - **LIFO** : Last in First Out
  - Examples
    - Stack of plates
    - Your turned in paper assignments
- What is a **Queue**?
  - Another sequential structure of **items**
  - that are **pushed at one end**
  - **Pulled** via the **opposite end**
  - **FIFO** : First In First Out
  - Examples
    - Queue for buying tickets
    - Traffic in one way single lane

Evaluating **prefix** expression

- Infix : (5+6)*3
  - * + 5 6 3
  - - + 2 * 3 4 / 16 ^ 2 3
- Algorithm (harder way)
  - Scan from right to left
  - Anytime you find an operator
    - Evaluate the operation using the previous 2 operands
    - Replace the <*operator*> <ộřêsắŋđ> < ộřêsắŋđ> with sêșụľʈ in the expression
  - Repeat the steps till only 1 operand is left

Evaluating postfix expression

- Evaluate
  - 2 3 1 * + 9 –
  - 5 3 + 6 2 / * 3 5 * +
- *Algorithm to evaluate (use **stack**)*
  1. Push the operands in a stack
  2. When you encounter a operator
     - Pop 2 operands
     - Perform the operation on operands
     - Push the result in stack
  - Repeat from 1 until expression is parsed
  - The last item in stack is the answer
    - There would be only 1 item left when done correctly
- * prefix expr can also be evaluated in this way (*just in **reverse***)

Evaluate

- Prefix
  - - * 5 + - 4 2 2 / 6 3
  - - * + 3 5 7 + / 4 2 1
  - - + 10 * 2 3 + 4 / 5 5
- Postfix
  - 1 2 + 3 4 + * 5 6 - / 7 +
  - 8 2 / 3 4 + * 5 1 + 2 / -
  - 9 8 4 2 1 ^ * / - 3 +

Answers

- Prefix
  - - * 5 + - 4 2 2 / 6 3
    - Ans : **18**
  - - * + 3 5 7 + / 4 2 1
    - Ans : **53**
  - - + 10 * 2 3 + 4 / 5 5
    - Ans : **11**
- Postfix
  - 1 2 + 3 4 + * 5 6 - / 7 +
    - Ans : **-14**
  - 8 2 / 3 4 + * 5 1 + 2 / -
    - Ans : **25**
  - 9 8 4 2 1 ^ * / - 3 +
    - Ans : **11**

Practice: Programming

- Write a function that takes a string and checks if it is a palindrome
  - Returns true if palindrome is found else false
  - Hints: string indexing, loops, if , comparing characters
- Count the number of vowels, consonants in a sentence.
  - Vowels : a,e,i,o,u
  - Consonant : everything else other than vowels
  - Ignore : spaces (‘ ’,) comma(,) dash(-),semicolon(;),colon(:)
- Find the most frequent word in a sentence
  - If more than 1 word has same frequency return the lexicographically smaller one.

Next week : Convert to Prefix and Postfix

- a + b - c * d + e ^ f
- (a + b) * c - (d - e) * (f + g)
- ((a + b) * (c + d) / (e - f)) + g
- a * (b + c) / (d - e)
- a - b / (c * d ^ e)
- (a + b) * (c + d) - e