Date: __/__/____

Balanced Tree

↳ Atleast Left subtree and Right subtree's height is 1 or 0.

↳ LS1 - LS2 = 0 or 1 (Good) ✓

# of elements

$O(\log_2 N) = X$ (running time)

$2^X = N$

X	N
1	2
2	4
3	8
4	16
5	32

Recursive Functions

↳ Functions that calls itself

↳ good substitute for a loop

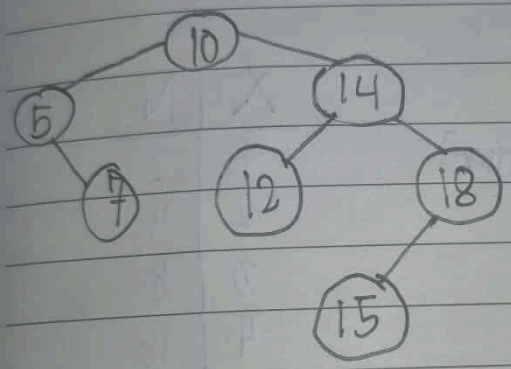
No Graph after 7/24/25

Date: __/__/____

Binary Search Tree (BST)

↳ insert first the root

SET A = {10, 5, 14, 12, 7, 18, 15}



OPERATIONS
- ↳ Insert
- ↳ Delete
- ↳ Member
- ↳ Min
- ↳ Max

elements < parent goes to the left

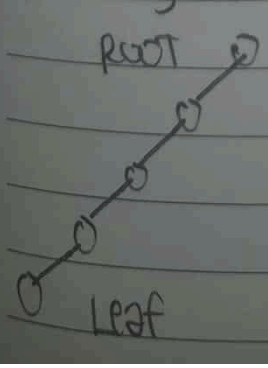
elements > parent goes to the right

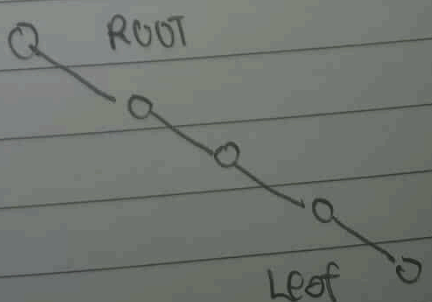
greater ≈ go right NO EQUAL !!

lesser ≈ go left

Skewed BST ⇒ (Better use linklist if its like this !)

(Ascending or Descending)





Date: __/__/____

Huffman's Algorithm

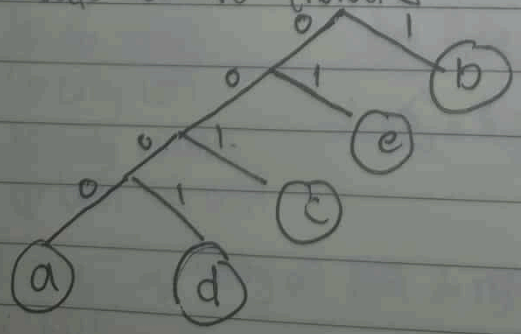
↳ technique in finding optimal prefix variable length code
↳ Implement Huffman's Algorithm using forest (collection of ~~trees~~)

Steps

1.) Initialize, each character is in a one-node tree by it's self.

2.) while (# of Trees > 1)
  a.) Select 2 trees with the smallest probabilities
  b.) Create a bigger tree from the two selected trees.
  c.) Make the sum of their probabilities the root of the new tree.

3.) Label left and right subtree with 0 & 1 respectively [The path from the root to any leaf represents the code of the characters]



Huffman
a = 0000
b = 1
c = 001
d = 0001
e = 01

Date: __/__/____

Application of Binary Tree

Fixed length - (000, 001, 010, 011, 100)

Variable length - (111, 11, 01, 101, 10) ⇒ EG: Morse Code

Average Code length - number of bits * probability of occurance

Symbol	Probability	Code 1	Code 2
a	.12	000	000
b	.40	001	11
c	.15	010	01
d	.08	011	001
e	.25	100	10

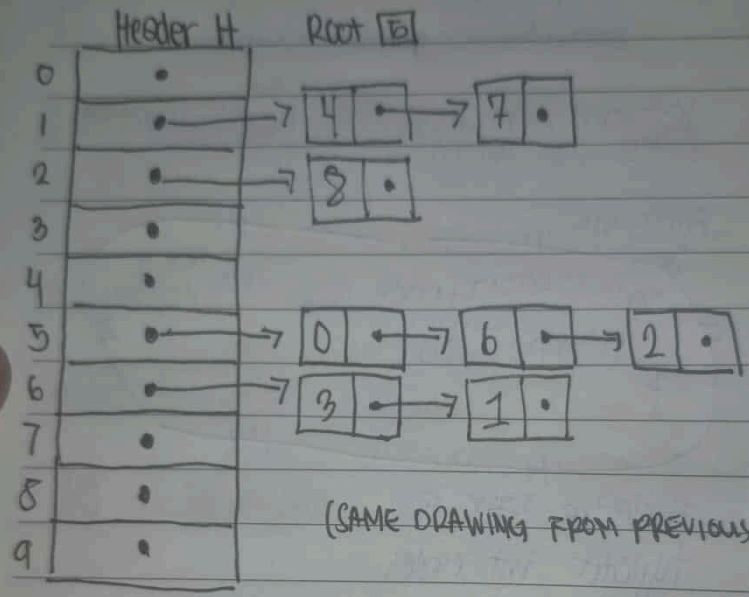
Message: (bade)

Code 1 ⇒ 001 000 011 100

Code 2 ⇒ 11 000 001 10

Date: __/__/____

2.) Representation of trees by list of children



(SAME DRAWING FROM PREVIOUS PAGE)

Binary Tree
- no child
- only left
- only right
- both has child

Avg Code length

$\sum$ = (# of bits / code) * probability of occurrence

Bintree => 2rray of left & right

0	1	-1
1	2	3
2	-1	4
3	4	-1

Page: ____

Date: __/__/____

2) Parent Pointer Representation

Tree T (DRAWING IN BOTTOM LEFT)

0	5
1	6
2	5
3	6
4	1
5	-1
6	5
7	1
8	2
9	-2

- left and right most child is not supported unless its sorted (Ascending).

Access via:

```
node PARENT(node n, Tree T){
    return T[n];
}
```

```
#define MAX 10
typedef int node;
typedef int Tree[MAX];
```

Date: __/__/____

CLOSED HASHING

2 Pass Insertion (delay insertion of synonyms)

1.) 1st pass: elements that are not synonyms

2.) 2nd pass: insert the synonyms

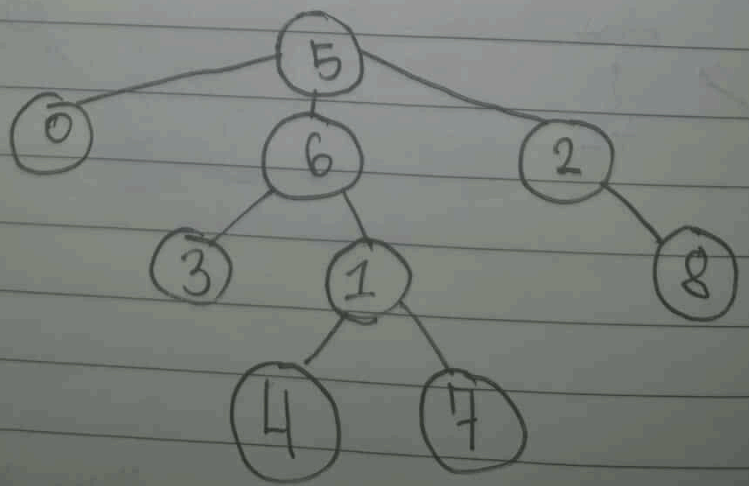
Average search length (ASL)

$$ASL = \frac{\sum \text{search length of each elements}}{\#\text{ of elements}}$$

TREES IMPLEMENTATION

1.) Parent pointer Representation

2.) Representation of trees by List of Children

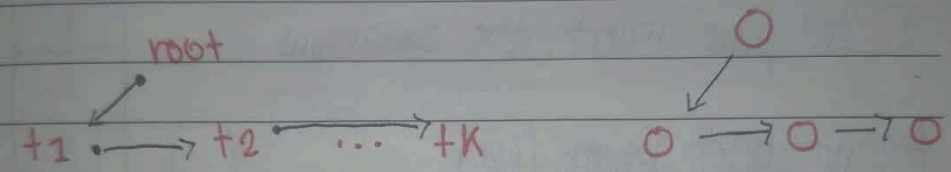


Date: __/__/____

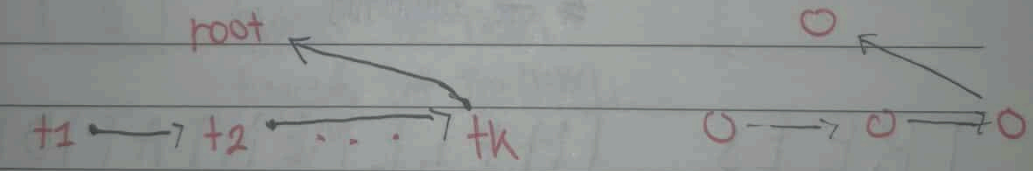
Traversal

- preorder => Root will be listed First
- postorder => Root will be listed Last
- Inorder

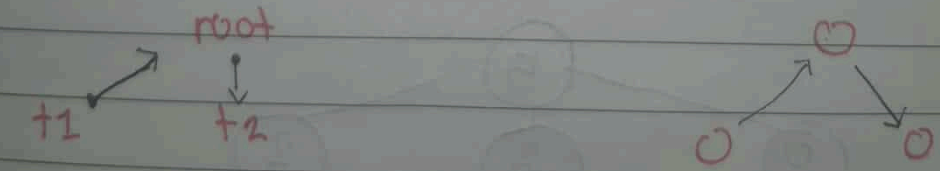
Pre-order



Post-order



In order



Infix

Prefix

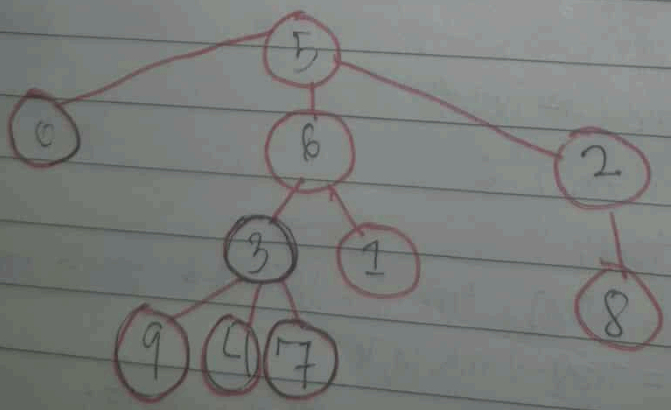
Postfix

Opima 7/10/25

Date: __/__/____

TREES

- Root - top most
- Children - has a parent node
- leaves - nodes without children
- decendant of 3 = 3 9 4 7
- ancestor of 3 = 3 6 5
- proper ancestor of 3 = 6, 5
- proper decendant of 3 = 9 4 7
- height = the length of the longest path from node to a leaf
- depth = the length of unique path from root to node.



Date: __/__/____

```
#define MAX 10

typedef struct node {
  int data;
  struct node *link;
} *Nodetype;
typedef Nodetype Dictionary[MAX];

void initDict(Dictionary *A){
  for(int i=0; i<MAX; i++){
    A[i] = NULL;
  }
}

Boolean member(int x){
  int y = hash(x){
  Nodetype trav;
  for(trav = A[val]; trav != NULL && trav->data != NULL;
      trav = trav->link){}
  return (trav != NULL)? TRUE : FALSE;
}

int hash(int x){
  return x % 10;
}

Closed Hashing ??
```

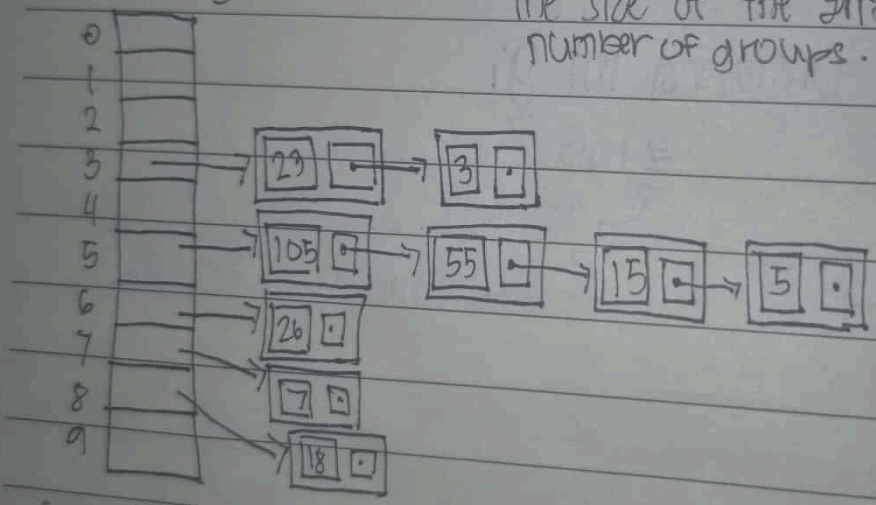
Date: __/__/__

Hashing

A.) Open Hashing

SET A = {3, 5, 26, 23, 7, 15, 55, 18, 105}

Dictionary A



The size of the array is the number of groups.

Grouping: one's digit => hash function if open hashing
=> hashing returns the group # of the group the element is member of

Date: __/__/____

2 kinds of right shift >>

1) arithmetic right shift pad with the sign bit

0 1

2) logical right shift pad w/ 0

char x; unsigned int y;

1 byte 4 byte

8 bits 32 bits

-128 to 127 possible value x 0 to 255

Date: __/__/____

Bit vector

Chatgpt said: "A bit vector (also called a bitset, bit array or bitmap) is a compact array of bits (0s and 1s), where each bit represents a boolean state (true/false, on/off). Instead of using 8 bits or 1 full byte for each boolean, a bit vector uses just 1 bit - making it memory efficient."

∈ Universal set is used as $2^{(U)}$ in bit vector, whereas in sets the universal set is used as indexes.

inclusive OR "|"
- 1 | 1 ≈ 1
- 1 | 0 ≈ 1
- 0 | 1 ≈ 1
- 0 | 0 ≈ 0

exclusive OR "|"
- 1 | 1 ≈ 0
- 1 | 0 ≈ 0
- 0 | 1 ≈ 0
- 0 | 0 ≈ 0

AND "&" ⇒ 1 & 1 ≈ 1
⇒ 0 & 0 ≈ 0

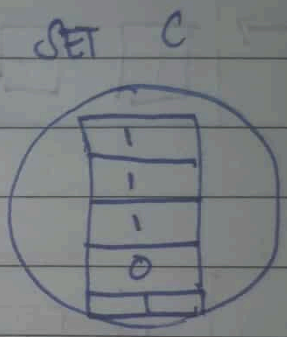
Compliment "~"
- 0~ ⇒ 1
- 1~ ⇒ 0

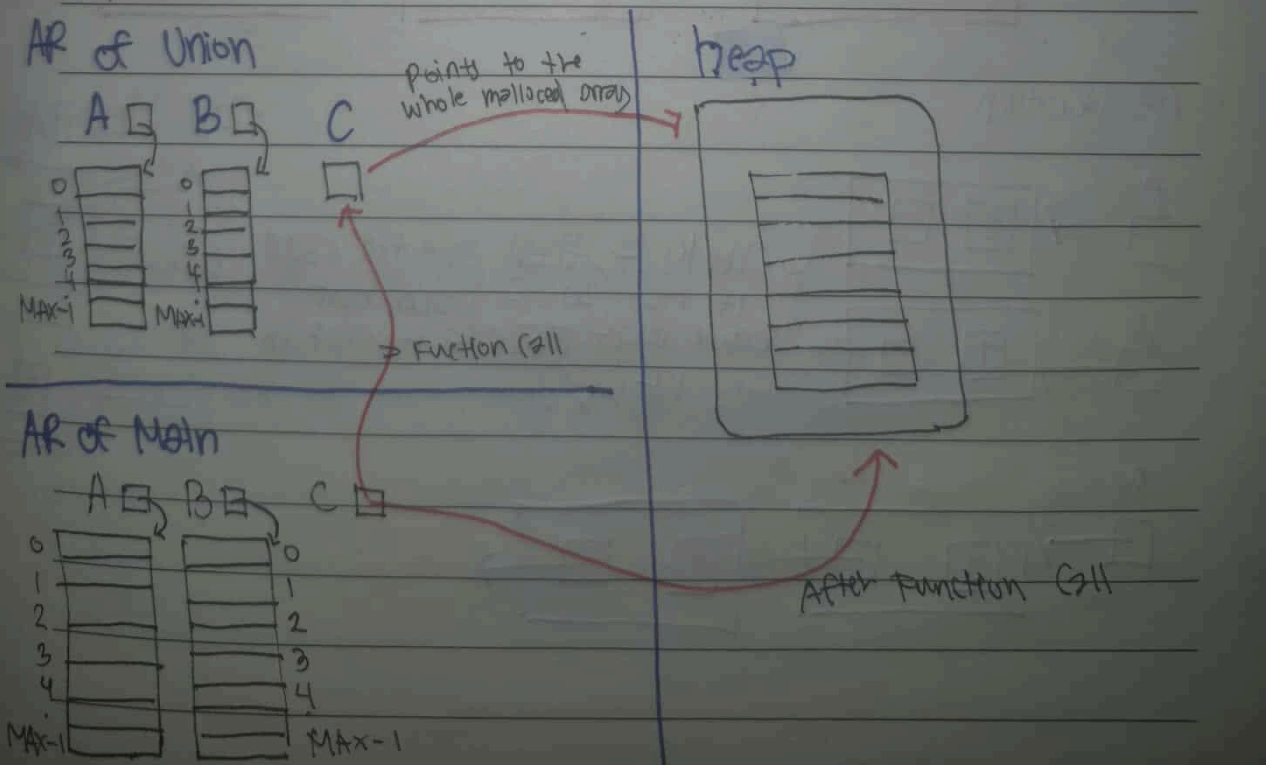
Page: ____

Date: __/__/____

Longer Version

```
SET *Union (SET A, SET B) {
  SET* C = (SET*) malloc (sizeof (SET));
  if (*C != NULL) {
    for (int i=0; i<MAX; i++) {
      if (A[i]== 1 || B[i] == 1) {
        (*C)[i] = 1;
      } else {
        (*C)[i] = 0;
      }
    }
  return C;
}
```

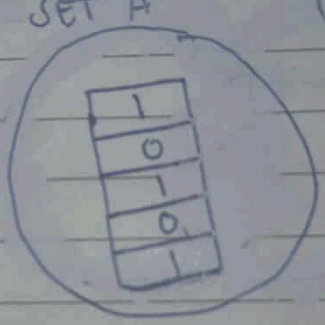




Date: __/__/____

List	vs	Sets
collection of elements which can exist in duplicates order is significant		collection of elements that are all unique (no duplicates) order is not important unless it is specified (sorted).

```
#define MAX 10
typedef int SET[MAX];
```



Inside the execution stack

```
SET A, B;
SET *C = Union (A,B);
```

Shorter Version

```
SET * UNION (SET A, SET B) {
    SET * C = (SET*) malloc (sizeof(SET));
    if (*C != Null) {
        for(int i=0; i < MAX; i++) {
            (*c)[i] = A[i] || B[i];
        }
    }
    return C;
}
```

Date: __/__/____

```
typedef struct {
  char elem[MAX]; // MAX is 8
  int rear;
  int front;
} Queue;

void init_Queue(Queue *Q){
  Q->rear = MAX-1;
  Q->front = 0;
}

void enqueue(Queue *Q, char data){
   if ((Q->rear+1) % MAX != Q->front){
      Q->rear = (Q->rear+1) % MAX; // Move first before inserting
      Q->elem[Q->rear] = data;
   }
}

void dequeue(Queue *Q){
   if ((Q->rear+1) % MAX != Q->front){
      Q->front = (Q->front+1) % MAX;
   }
}
```

Page: ____

Date: __/__/____

Queue → Circular Arrays Implementation

- Fixed size
- cyclic storage

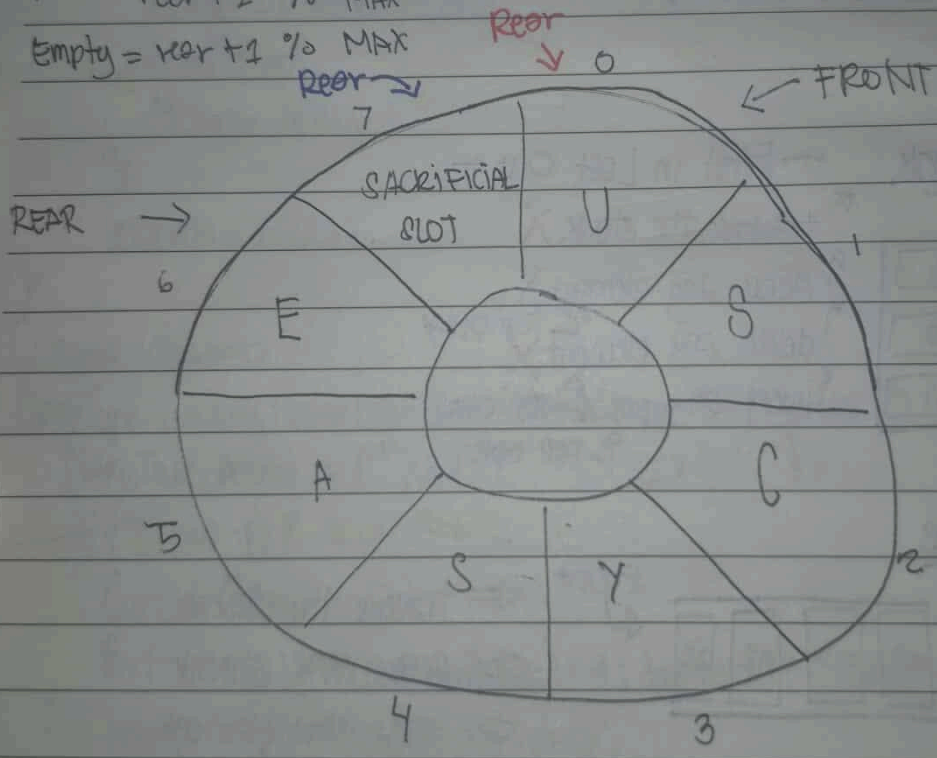
1 element = front is equal to rear

FULL = rear + 2 % MAX

Empty = rear + 1 % MAX

FRONT → Points to the first valid elements

REAR → Points to the last element where insertion to the next is possible or list is full.



When List is !

rear + 2 % MAX = 0

6 + 2 % 8 = 0 ✓ FULL

rear + 1 % MAX = 0

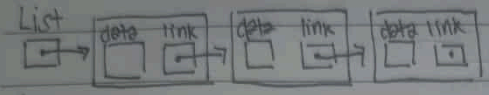
7 + 1 % 8 = 0 ✓ Empty

Rear = FRONT

1 element in the list or array.

Date: __/__/____

ADTs List

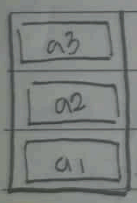


- traverse the List ✓
- access any elem ✓
- delete any elem ✓

List Stack
- insert ⇒ (push) / (insertTop)
- delete ⇒ (pop) / (deleteTop) ← if it exist

Top = returns the top element of stack.

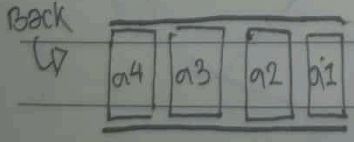
Stack — First in Last Out



- traverse the stack X
- access any element X ← TOP ONLY
- delete any element X
- insert any pos X ← TOP ONLY

← TOP ONLY

Queue



- ⇐ traverse the Queue
- ⇐ access any element
- ⇐ delete any element

- access top/front only
- insert rear only
- delete front only

Queue Operation
- enque (insert) ⇒ insert rear
- deque (delete) ⇒ delete front
- front (access) ⇒ return front element

FIFO - First in First Out

Date: __ / __ / ____

CURSOR BASED (Helper Functions)

```
int allocSpace (VirtualHeap *VH){
  int temp = VH->Avail;
  if (temp != -1){
      VH->Avail = VH->Nodes[temp].link;
  }
  return temp;
}
void deallocSpace (VirtualHeap *VH, int index){
  VH->Nodes[index].link = VH->Avail;
  VH->Avail = index;
}
void initLIST (VirtualHeap *VH){
  VH->Avail = MAX-1
  for(int i=0; i<MAX; i++){
     VH->Nodes[i].link = i-1;
  }
}
```

initLIST => initializes the cursor based list to be connected

allocspace => returns the index of the next available space and -1 if the array or list is full.

deallocspace => Makes the node or index available for the next usage of allocSpace.

Page: ____

Date: __/__/____

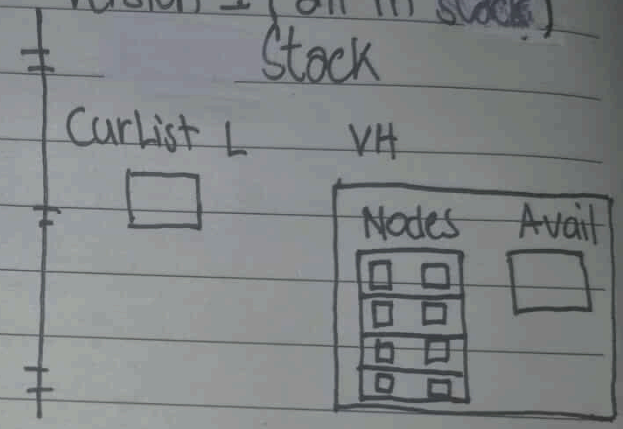
Cursor Based Visualization in Memory

```
typedef struct {
    char elem;
    int link;
} NodeType;

typedef struct {
    NodeType Nodes;
    int Avail;
} VirtualHeap;
```

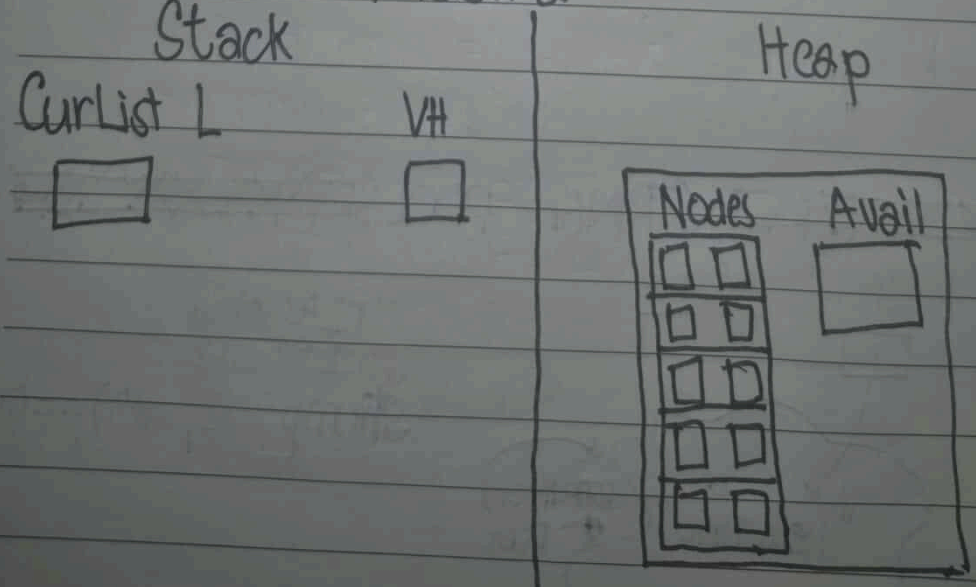
```
#define MAX 10
```

Version 1 (all in stack)




```
typedef int CurList;
```

Version 2

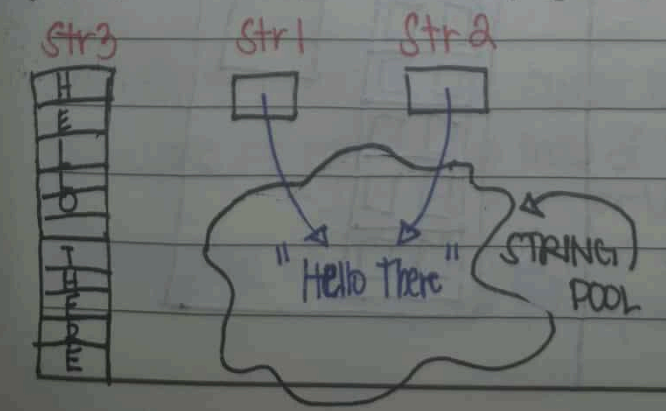


MEMORY

Date: __/__/____

- **Stack**: Fast, short-term, auto-managed memory for local variables.
- **Heap**: Flexible, long-term, manually-managed memory for dynamic data.

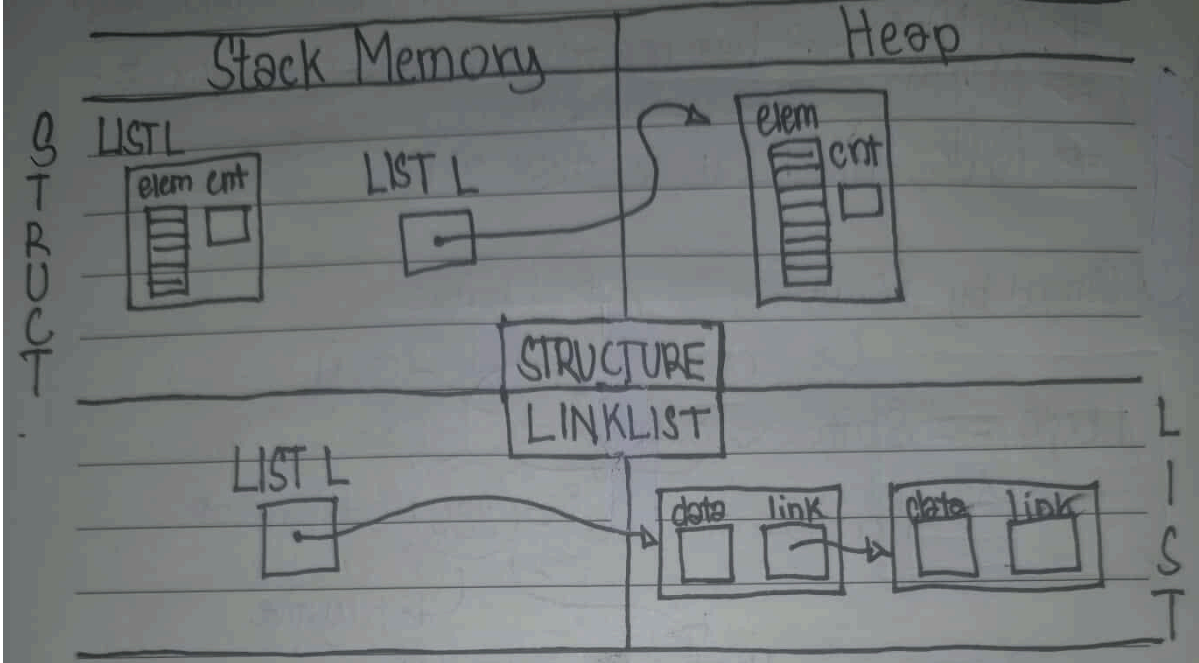
STRING POOLS



str1 ≈ str2
str1 ≈ str3
strcmp(str1, str3) == 0

Notes on cursor board??
aquina 6/30/25

Visualization in Memory



Note:

⇒ When using malloc always check if it is successful for good practice.

⇒ Data Segment is the part of a program's memory where Global variables, static variables (inside or outside functions) are stored.

⇒ String Pools

- is a shared memory area that stores only one copy of each unique string literal. When multiple variables are assigned the same string literal. they point to the same memory address in the pool saving memory.

```
char *str1   = "Hello There"
char *str2   = "Hello There"
char str3[6] = "Hello There"
```

Date: __/__/____

Running Time (Execution)

- ⇒ O(1) ⇒ constant time, insertFirst in Linklist and ⇦ insertLast in arrays
- ⇒ O(N) ⇒ time depends on number of elements.
- ⇒ O(N^2) ⇒ Searching Algorithm
- ⇒ O($\log_2 N$) ⇒ (to be discussed)

Singed by default ⇒ 1 byte

↳ char x; ⇒ (0, 1)

1 byte == 8 bits ≈ 0000 0000

total values or combination = 2^8 = 256 combination

= Positive = 1 to 127

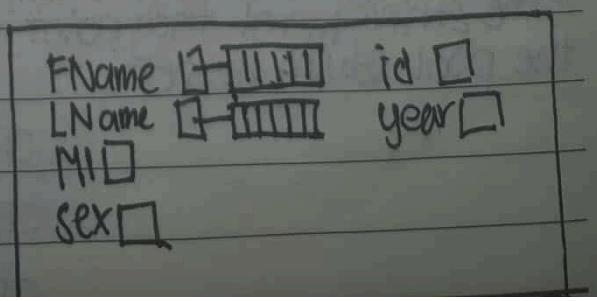
Negative ⇒ -1 to -128

0 is from positive 1 to (128-1)

(Sit In) JUNE 26 2023 PROG 2 CLASS. BY. Mr. Granito

```
#define MAX 20
typedef struct{
    char FName[MAX];
    char LName[MAX];
    char MI;
    bool sex;
    int id;
    int year;
} student;
```

student[1].id
(student+1)->id
(*(student+1)).id
} does the same thing



Date: __/__/____

✓ Structure Definition and Declaration at the same time is Okay().

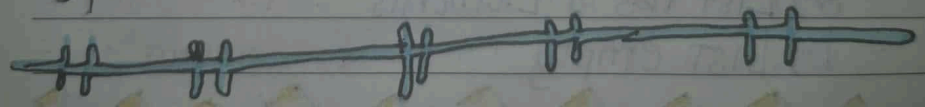
e.g.

```
struct Person {
    char name[20];
    int age;          // p1 is declared //
} p1; // The structure is defined
```

✗ Structure Definition after Declaration does not work and is not Okay().

e.g.

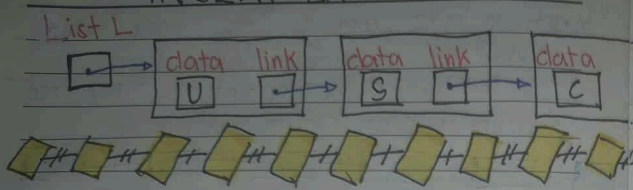
```
struct Person p2; // struct Person is not yet defined //
struct Person {
    char name[20];
    int age;
};
```



- unsigned ID ~ unsigned int ID
- int ID ~ signed int ID

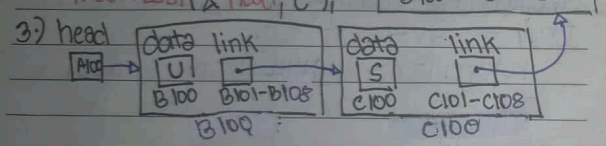
"INSERT LAST"

Date: __/__/____



1.) Void insertLast(List *L, char ch);

2.) List head = NULL;
insertLast(&head, 'U');
insertLast(&head, 'S');
insertLast(&head, 'C');



3.) head

4-5.) List temp = (List)malloc(sizeof(struct node));
temp → data = ch;
temp → link = NULL;
if(*L == NULL){
*L = temp;
} else {
List *ptr;
for(ptr = L; (*ptr) → link != NULL; ptr = &(*ptr) → link) {}
(*ptr) → link = temp;
}

// write a function
// PPN does not use look-ahead

*ptr != NULL ← NOT USING LOOK AHEAD

Note: PPN

*ptr = temp Attached to the NULL (END).

Page: ____

Date: __/__/____

steps

1.) Write the function header
2.) Write an appropriate function call, dec
all variables in the call & initialize if nee
3.) Assure that the function call is in main(
draw the execution stack. For each variable
draw a box & label it with name, value, a
address.
4.) Based on the execution stack, write t
code of the function. A for loop that lets
ptr to point to ListL, then move ptr unti
will point to the link pointer that is NULL.
5.) Using test cases, checks the code for co
a.) List has 3 Elements
b.) List empty

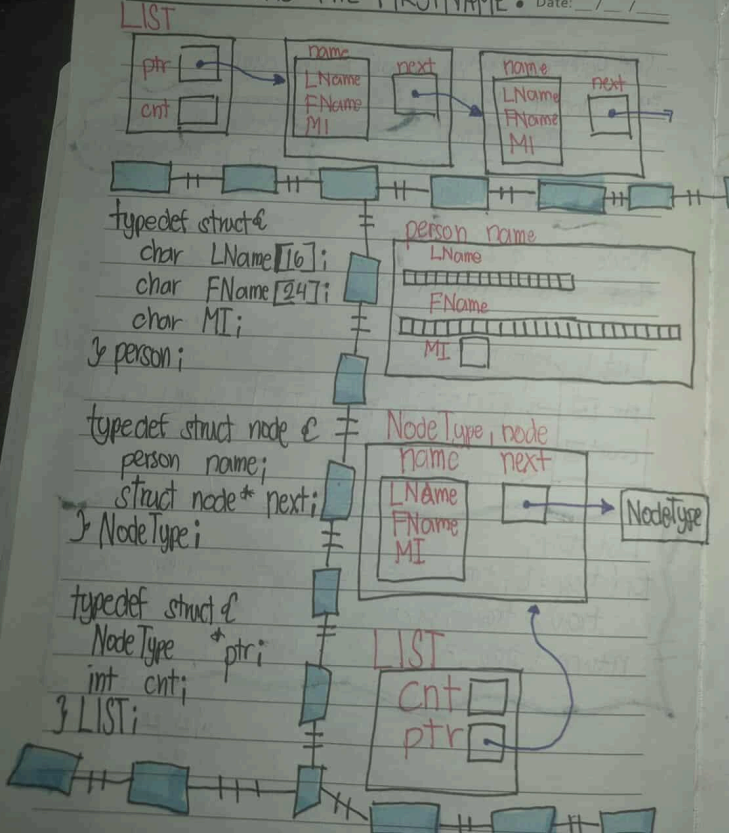
typedef struct node {
char data;
struct node *link;
} *List;

"I MISS PYTHON" - Char

Page: ____

FIND THE FIRSTNAME!

Date: __/__/____

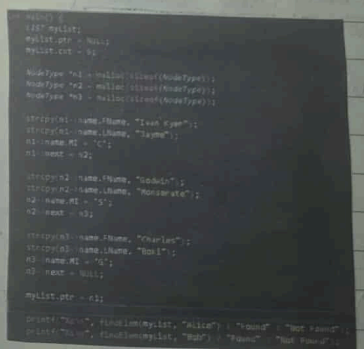


Date: __/__/____

"In defining lowest scope to largest scope"

- lowest scope is name(variable) or person struct
- In NodeType; define node first to self reference
- Define person before NodeType

```
bool findElem(LIST L, char fname[]);
bool findElem(LIST L, char fname[]){
  NodeType *trav;
  for(trav = L.ptr; trav != NULL &&
strcmp(trav->name.FName, fname) != 0;
  trav = trav->next){}
    return (trav != NULL)? true : false;
```



Date: __/__/____

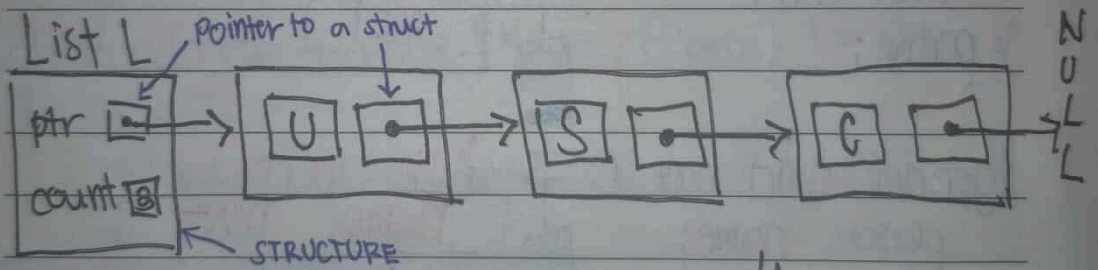
- Self Reference structure requires a tag name

eg:
```
typedef struct cell {
    char elem;
    struct cell * next;
} Node, *List;
```

define first before using in self referencing structs

Node - is a struct

*List - is a pointer to a struct




```
List trav;
for(trav = L; trav != NULL && trav->elem != X;
    trav = trav->next){}
return (trav != NULL) ? true : false;
```

6/23/25