16/6/16

Unit 1:

Data Abstraction:

- Referring essential features and hidding all other implementation details.

Data Type:

- It is a collection of objects (⇒ Properties/attributes/fields) and the set of operations performed on those objects.
- In C++ and Java structures, classes are used to represent the data type.

ADT (Abstract Data Type):

- The specification of objects and their operations are separated from object representations and implementation of the operations. Egs: stacks, queues, linked list, trees.

Egs: Ex1

```
Struct student
{
  int ht no;
  Char Sname[30];
  float Sub[5];
  float tot, avg;
Public:
  Void get Student Details ( );
  Void find Total ( );
  Void find Avg ( );
  Void display Student Details ( );
};
```

Ex 2:

```
Class stack.
{
   int top;
   int S[10];

Public:
     Void Push(int x);
     void
      int Pop();
      Void display();
    };
```

~~Excamples~~

Performance Analysis

important factors are.

1. Time taken by Algorithm. (Refered to as Time Complexity)
2. Space occupied. (Space Complexity).

Time complexity

Algorithm Execution time depends on.

- Type of Machine (Type of processor)
- Type of Instruction set.
- Type of compiler.
- Type of OS (single tasking processor / multi tasking Processor)

- Hence we do not ~~Exa~~ actually count the time taken by the algorithm instead we measure the time Complexity as count of no. of instructions

get executed before the Program/algorithm generates the output.

finding the time complexity:

Ex1: Sample code.

```
int i=1, sum=0;          → 1.
for (i=1; i<=n; i++)     → (n+1)
{
sum = sum + i;           → n.
};
pf ("%d", sum);          → 1
```

2n+3.

Ex2

```
for (i=1; i<=m; i++)          → m+1
{
for (j=1; j<=n; j++)          → m(n+1)
{
pf ("%d", A[i][j]);           → m.n
}
}
```

m+1 + mn + m + mn.

2mn + 2m + 1

ex3:

```
for (i=1; i<=n; i++)                    ---> (n+1)
{                                       ---> n(n+1)
                                                = n^2+n
for (j=1; j<=n ; j++)
  {                                     ---> n * n = n^2
   c[i][j]=0 ;
for (k=1; k<=n; k++)                    ---> n^2(n+1)
                                                = n^3+n^2
  {
  C[i][j] = C[i][j] + (A[i][k] * B[k][j]);  ---> n^2(n)
                                                = n^3
    }
  }
}
```

$2n^3 + 3n^2 + 2n + 1$

7/6/16

Asymptotic Notations.

These Notations are used to represent time complexity of the algorithm.

1) Big O (Big oh)
2) omega (Ω)
3) Theta (Θ)

Big O Notation:

- The function $f(n) = O(n)$ if and only if their exists +ve constants c, n_0 such that $f(n) \leq c \cdot g(n)$ $\forall$ $n \geq n_0$

Examples:

1) $f(n) = 3n+2$

Let $g(n) = n$.

c ↓ ④ , n_0 ↓ ②

$$f(n) \leq 4 \cdot g(n) \quad \forall \quad n \geq 2$$

$$f(n) = O(g(n)) = O(n)$$

2) $f(n) = 100n + 6$.

$g(n) = n$.

$f(n) \leq c \cdot g(n) \quad \forall \quad n \geq n_0$.

$c = 101$ $\quad$ $n_0 = 6$ $\quad$ $f(n) = O(n)$

3) $f(n) = 6 \times 2^n + n^2$.

$g(n) = 2^n$ $\quad$ $c = 7$

$f(n) \leq 7 \cdot g(n)$.

$\leq 7 \cdot 2^n \quad \forall \quad n \geq 4$.

$n_0 = 4$ $\quad$ $f(n) = O(2^n)$

~~4) $100 = 10n^2$~~

4) $f(n) = 10n^2 + 4n + 2$.

$g(n) = n^2$ $\quad$ $c = 11$

$f(n) \leq 11 \cdot g(n)$

$10n^2 + 4n + 2 \leq 11 \cdot n^2 \quad \forall \quad n \geq 5$

$n_0 = 5$ $\quad$ $f(n) = O(n^2)$

Omega Notation:

The function $f(n) = \Omega(g(n))$ if and only if there exists +ve constants c, n_0 such that $f(n) \geq c \cdot g(n) \; \forall \; n \geq n_0$

Examples:

1) $f(n) = 2n^2 + 4n + 1$

$g(n) = n^2$

$f(n) \geq 2 \cdot g(n) \; \forall \; n \geq 2$

$2n^2 + 4n + 1 \geq 2n^2 \; \forall \; n \geq 0$. $f(n) = \Omega \cdot (n^2)$

Theta Notation:

The function $f(n) = \Theta(g(n))$ if and only if there exists +ve constants c_1, c_2, n_0 such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \; \forall \; n \geq n_0$

Example:

1) $f(n) = 10n^2 + 4n + 2$

$g(n) = n^2$

$c_1 = 10$ (can be either 10 or 9)

$c_2 = 11$

$10 \cdot n^2 \leq 10n^2 + 4n + 2 \leq 11 \cdot n^2 \quad \forall \; n \geq 5$

$f(n) = \Theta \cdot (n^2)$

Space Complexity :-

- The amount of memory required by the program for its Execution.
- It is actual denoted as $S(P) = C + Sp$.

where :- C is fixed Portion (fixed size).
Sp is variable size.

Examples for fixed Part :- Instructions, Variables, identifiers.

Sp → variable size → This is the memory need for variables that depend on individual Problem instance.
Ex :- Recursion stack.

21/6/16

Linked list

```
void creat()
{
struct node * newnode, * Prenode;
int x; char choice;
do
 {
 Pf ("Enter the Number ");
 Sf ("%d ", &x);
newnode = (struct node *) malloc (size of (struct node));
newnode → data = x;
newnode → link = NULL;
                if (root == NULL)
                    root = newnode;
                else
                Prevnode → link = newnode;
```

```
Prevnode = new node;
pf(" do you wish to continue (y/n)");
getchar();//fflush();
choice = getchar();
}
while (choice == 'y');
}.

void display()
{
struct node *temp = root;
while (temp != NULL)
{
pf ("%d ", temp->data);
temp = temp->link.
}
}

void main()
{
create()
display();
length();
}
```

22/6/16

```
int
{
int
c
}
```

Line

```
voic
{
int
whi
{
```

```
If (flag =
pf ("
else
pf
}.
```

22/6/16

```
int length()
{
int l=0; struct node *temp=root;
while (temp != NULL)
  {
    l++;
  temp = temp->link;
  }
  return l;
}
```

Linear Search Using Linked List:

```
void Search (int key)
{
int flag=0; Struct node *temp=root; int pos=0;
while (temp != NULL)
{
  pos ++;
   if (temp -> data == key)
      {
        flag=1; break;
      }
    temp = temp -> link;
    }.
if (flag == 1)
  pf ("found at %d", pos);
  else
    pf ("Not found");
  },
```

```
Void search (int k, int *f, int *p)
{
Struct node *temp=root;
while (temp != NULL)
{
(*p)++;
if (temp->data == key)
{
*f = 1; break;
}
temp= temp->link;
}
}

Void append (int x)
{
Struct node *temp, *newnode; temp= root;
newnode = (Structnode *) malloc (sizeof (struct node));
newnode->data = x;
newnode->link = NULL;
while (temp->link != NULL)
{
temp = temp->link;
}
temp->link = newnode;
}
```

```
/// append (insert) at any position

Void append (insert) (int pos, int x)
{
Struct node *temp, *newnode; temp=root; int i=2;
newnode = (Struct node*) malloc (sizeof( struct node));
newnode -> data = x;
newnode -> link = NULL;
  if (pos == 1)
     {
     newnode -> link = temp;
     root = newnode;
     }
else
   {
while (i< pos && temp -> link != NULL)
     {  i ++;
ode));     temp = temp -> link;
     }
newnode -> link = temp -> link;
 temp -> link = newnode;
}
}
```

23/6/16

// deletion of node based on position.

```
void delt (int pos) {
Struct node * temp = root , * temp2 , * temp3 ;
int i = 2 ;
if (Pos == 1) {
    temp2 = root ;
    root = root → link ;
    free (temp2) ;
    }
else {
  while (i < pos && temp → link != NULL) {
        i++ ; temp = temp → link ;
        temp3 = temp → link ;
  } if (temp → link != NULL) {
  temp → link = temp → link → link ;
  free (temp3) ; } else { pf ("invalid pos") ;
  }
  }
```

// deletion of node based on value given

```
void search (int k ; int * f ; int * p) {
  Struct node * temp = root ;
  while (temp != NULL) {
      * p++ ;
```

```
if (temp->data == k) {
        *f++;
        break;
    }
  temp = temp->link;
  }
  }
void main(){
  search (key; flag, &pos);
  if (flag == 1) {
      delet (pos);
    }
else{
pf("can't delete");
  }.
```

// display using recursion: (in forward)

```
void display(struct node *temp){.
  if (temp != NULL){
  pf("%d", temp->data); }
  display (temp->link);
  }
  }
```

* interchange these statements will get display reverse.

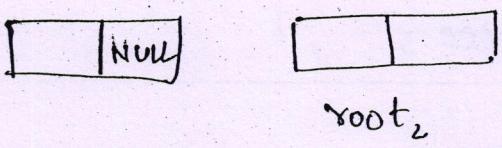
```
void main(){
  struct node *root = NULL;
        create (&root);
        display(root);
        }.
```

```
void Create (Struct node ** root) {
    .
    .
    .
if (* root == NULL) {
        * root = new node;
    }
}
```

// display using recursion (in backward):

```
void display reverse (St
```

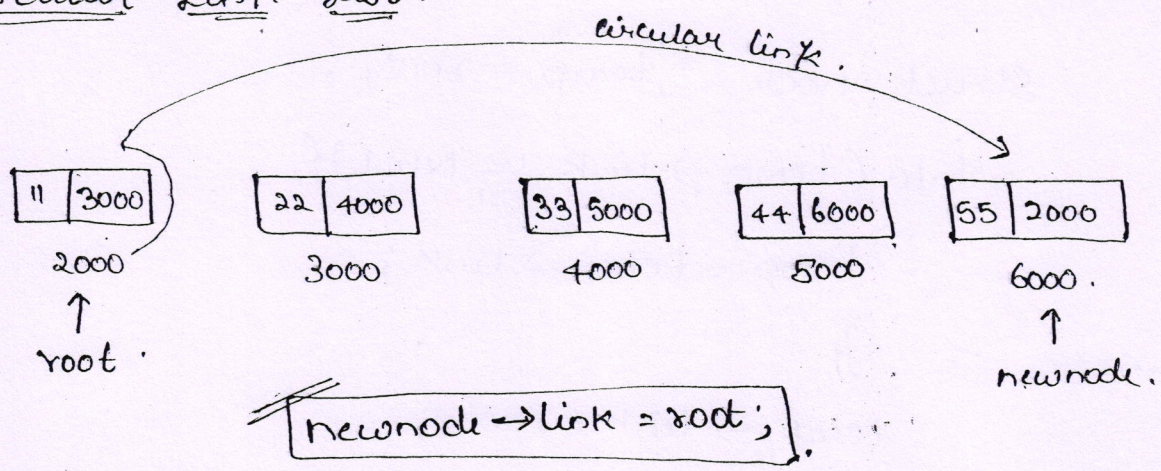
```
// concatenate two linked lists;
void concat (struct node * root1, struct node* root2)
{
  struct node * temp = root1;
  while (temp -> link != NULL){
      temp = temp -> link;
      }
    temp -> link = root2;
void main () {
  struct node *root1 = NULL,
              * root2 = NULL;
  create (& root1);
  display (root1);
  create (& root2);
  display (root2);
concat (root1, root2);
  display (root1); }
```



root2

28/6/16

Circular Link List :-




```
newnode->link = root;
```

~~void creat.~~

```
#include <stdio.h>.
Struct node{.
 int data;
 Struct node * link;
 }* root = NULL;

void creat()
 {
Struct node . * newnode, * Prevnode .
 int x , char Choice ;
 do
  {
  Pf ("Enter the number");
  Sf (" %d ", & x);
 newnode = (Struct node*) malloc ( Size of (Struct node));
 new node → data = x ',
  newnode → link = NULL',
```

```
if (root == NULL)
        root = newnode;
else
Prevnode -> link = newnode;
 Prevnode = newnode.
pf("DO you wish to Continue (y/n)");
get char();
Choice = get char();
} while (Choice == 'y');
  newnode -> link = root;
}
```

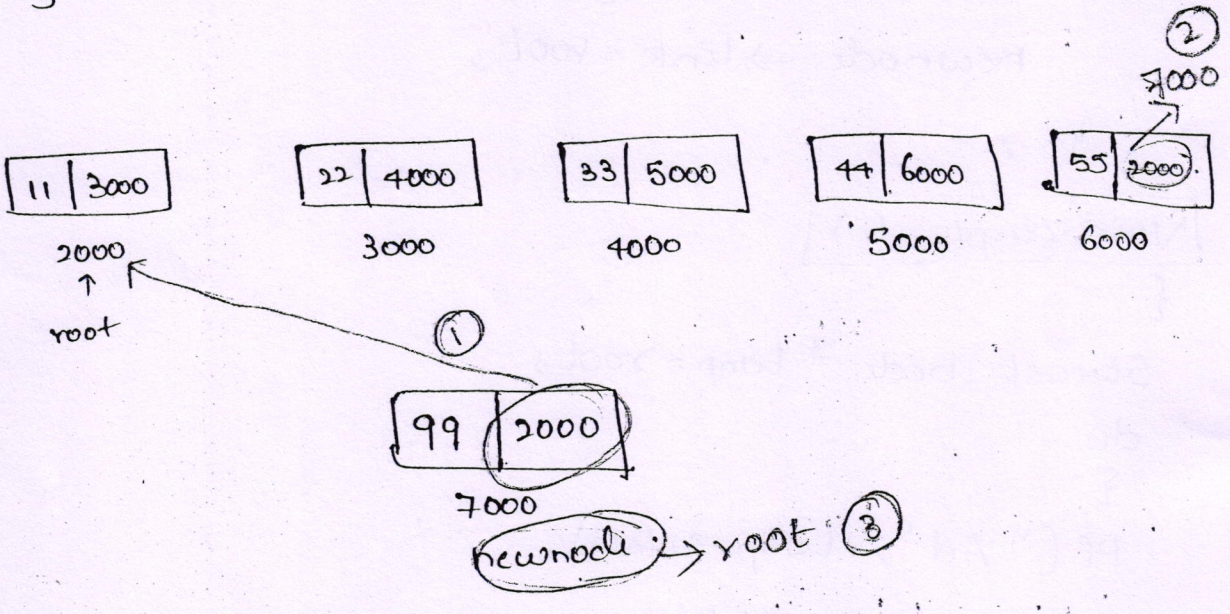
```
Void display()
{
Struct node *temp = root;
do
 {
 pf("%d", temp -> data);
   temp = temp -> link;
 }
while (temp != root);
```

```
Void insert()
temp = root , i = 2
newnode = (struct node*) malloc (Size of (struct node));
newnode -> data = x;
 newnode -> link = NULL;
if (POS == 1)
{
newnode -> link = root; ①
```

```
while (temp->link != root).
    {
    temp = temp->link;
    }
temp->link = newnode';
```

(2)

```
root = newnode; (3)
}
```




```
// at position (Eg: 8 (or) 4...)
else {
  while (i< pos && temp -> link != root)
  {
    i++;
    temp = temp -> link';
  }
  new node -> link = temp -> link;
    temp -> link = newnode';
  }
}.
```

⇒ Void delete (int pos){.

Struct node *temp = root; *temp1;

int i=0; int i=2;

while (temp → link != root){.

i ++;

temp = temp → link;

}

if (pos == 1) {.

{

if (temp → link == root)

{

root = NULL;

else }.

while (temp → link != root).

{

temp = temp → link;

}

temp → link = root → link; ✓

root = root → link;

}

}

③ [at any position]

else if{

while (i < pos && temp → link != root){.

✓ i++;

temp = temp → link;

temp1 = temp → link;

}

temp → link = temp → link → link; ✓

free (temp1);

}

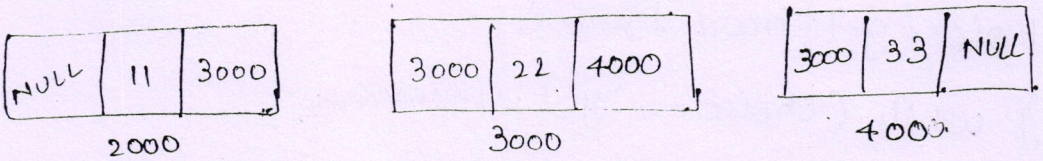
```
else {
    Pf ("Invalid pos");
    }
    }.
```

```
#in
St
{
S
}
```

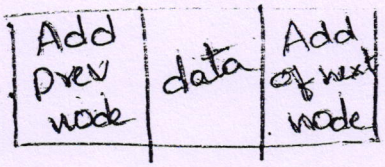
→ Voic
{
Stic
int
{
Pf
sf(
newr
newr
newn
newn

el

Double Linked List



root →




```
#include <stdio.h>
Struct node.
{
 Struct node *left;
    int   data;
 Struct node *right;
 } * root = NULL; * Last = NULL;

→Void create ( )
 {
Struct node. *newnode, * Prevnode, * last;
int x, char choice;
do {
 Pf ("Enter the Number");
 Sf ("%.d", &x);
newnode = (Struct node *) malloc (sizeof (Struct node));
newnode → data = x;
newnode → left = NULL;
newnode → right = NULL;
        if (root == NULL)
            root = newnode;
  else{ Prevnode → right = newnode;
  new
  Prevnode → left = Prevnode; }.
```

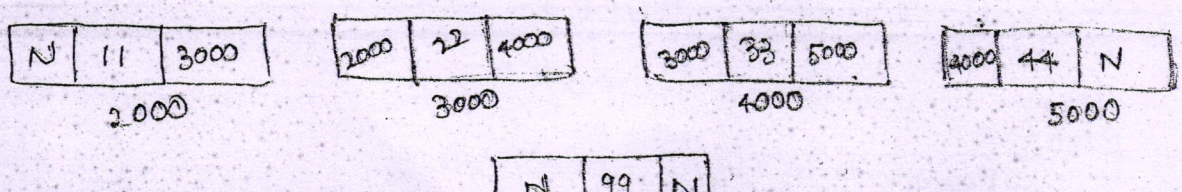
```
prev = newnode;
Pf ("Do you wish to continue (y/n)");
getchar();
choice = getchar();
} while (choice == 'y');
last = newnode;
}
```

⇒ Void display ( ) for forward display.

```
{
struct node *temp = root;
while (temp != NULL)
{
Pf ("%d", temp → data);
temp = temp → right;
}
}
```

for backward display.

```
Struct node *temp = last;
while (temp != NULL)
{
Pf("%d", temp → data);
temp = temp → lft;
}
}
```



30/6/16

⇒ Void insert (int pos, int x)

{.

Struct node *temp, *newnode, temp=root ; int i=2;

newnode = (struct node *) malloc (sizeof (struct node)) ;

newnode → left = NULL ;

newnode → data ;

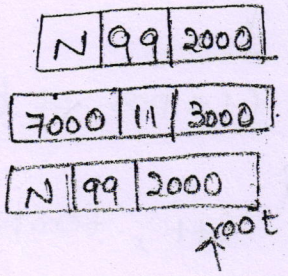
newnode → right = NULL ;

If (pos ==1) {.

newnode → right = root ;

root → left = newnode ;

root = newnode ;



}

else {

while (i< pos && temp → right != NULL) {.

i++ ;

temp = temp → right ;

}

newnode → right = temp → right ;

newnode → left = temp ;

if (temp → right != NULL)

temp → right → left = newnode ;

else.

last = newnode ; (for updating the last newnode)

temp → right = newnode ;.

}.

N

)0

```
→ void delete (int pos)
  {
  struct node *temp; int i=2;
  if (Pos==1)
    {
    root = root → right;
    root → left = NULL;
    }.
while (i<pos && temp → right != NULL).
    {
      i++; temp = temp → right;
    }
if (temp → right != NULL)
  {.
  temp → right = temp → right → right;
    if (temp → right != NULL).
      temp → right → left = temp;
      else
          last = temp;
    }
else
    {
    pf("invalid position");
    }
```