Polynomial Operations using singly Linked list

LECTURE 16: Nov. 27th, 2021(10.30 am to 12.30pm)

Polynomial Representations

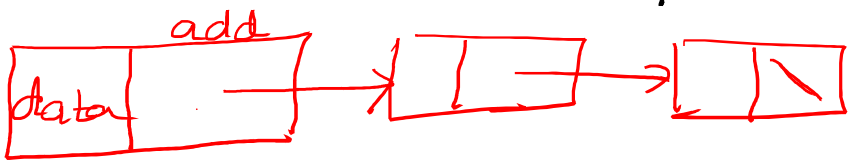
$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \cdots + a_0 x^{e_0}$$

$A(x) = 3x^2 + 1x^0$

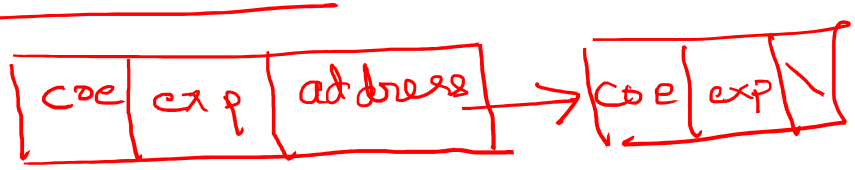
$a_i \Rightarrow$ Nonzero coefficients

$e_i \Rightarrow$ nonnegative integer exponents

i.e. $e_{m-1} > e_{m-2} > \cdots > e_0 \geq 0$

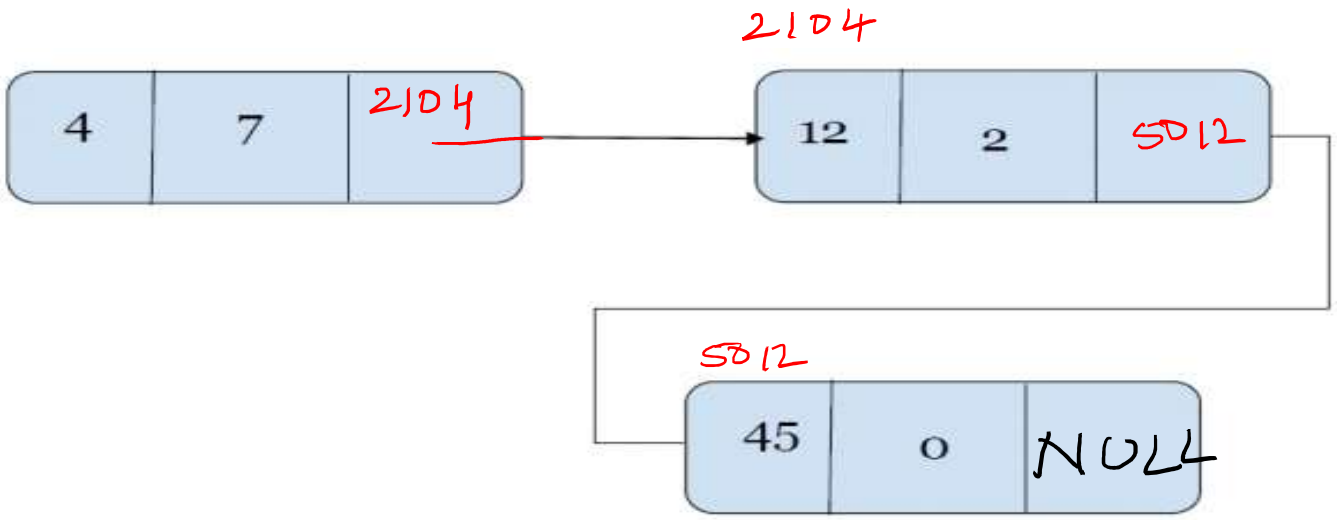


Polynomial class



Polynomial represention:

Polynomial : $4x^7 + 12x^2 + 45x^0$



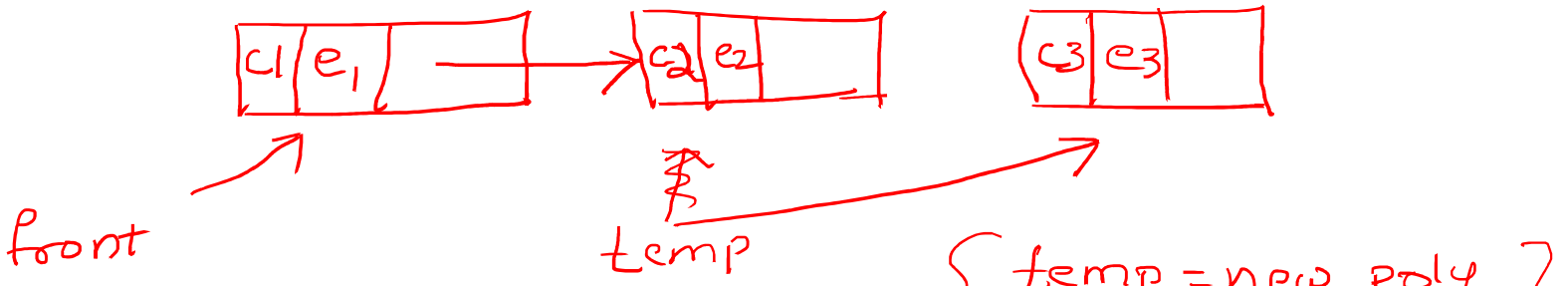

```
class poly{
        int coe;
        int exp;
        poly * link;
public:
    ===
    ===
};
```

To read a Polynomial list and return the address of the first node

poly *front = NULL;

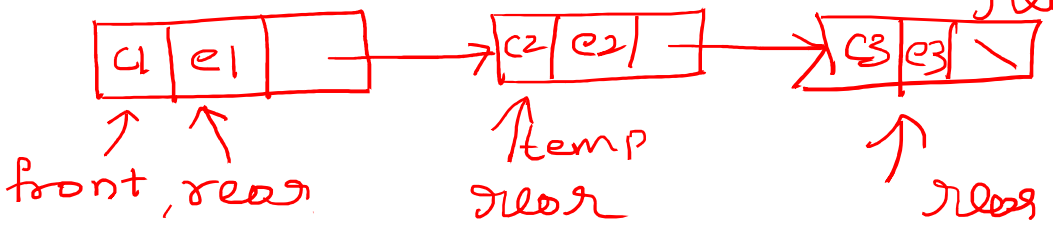
n → terms in the polynomial } read from user

c, e

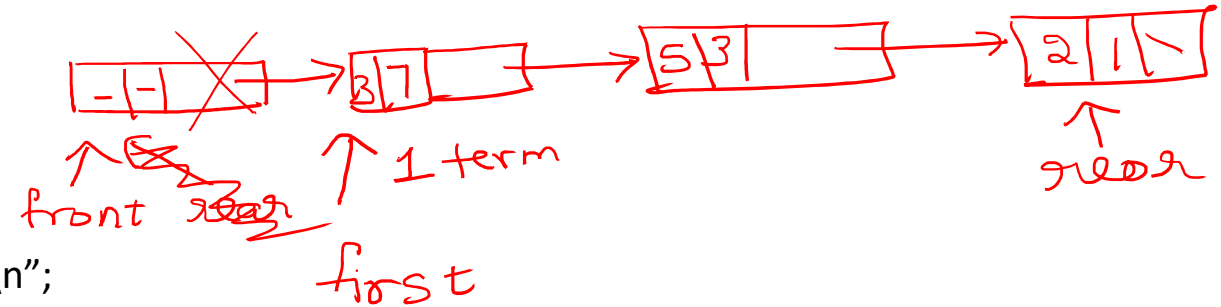


front → link = temp

{ temp = new poly
rear → link = temp
rear = temp; }



Function to read a Polynomial list and return the address of the first node

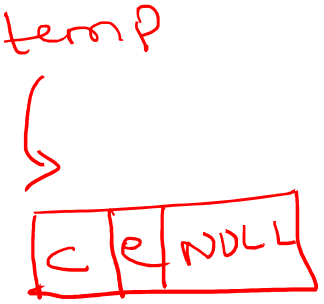



```
poly * poly:: read()
{
        int n,c,e;
        cout<<"how many terms?\n";
        cin>>n;

        poly *front=new poly; //dummy node
        poly *rear=front; //using q concept
        for (int i=0;i<n;i++)
         { cout<<"Enter coe and exp";
           cin>>c>>e;
           rear=attach(c,e,rear); //to insert a node at the rear i.e. at the end
        }
        poly *first = front->link;//actual first node
        delete front;
}
```

Attach function to insert a node at the next postion

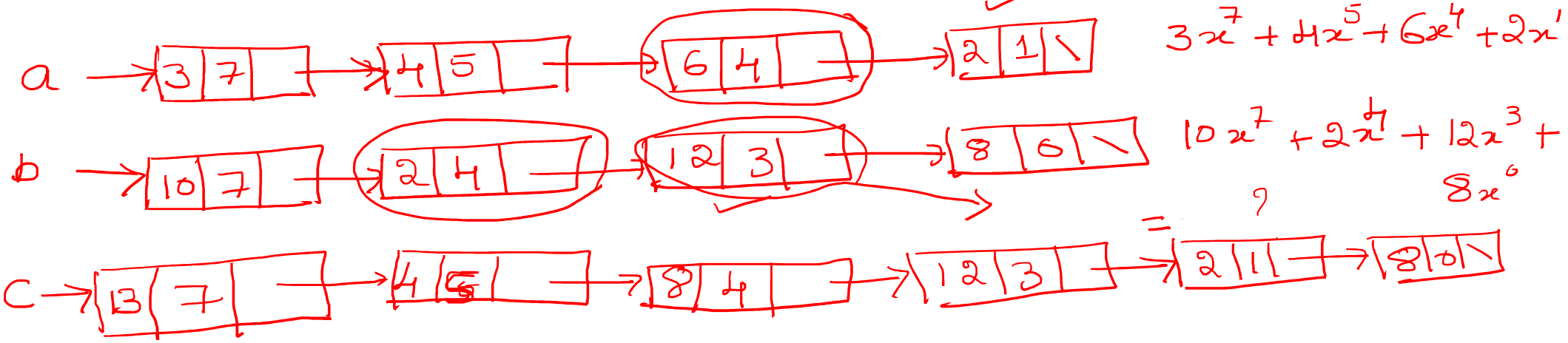
```
poly *poly::attach(int c, int e, poly *rear)
{
    poly *temp = new poly;
    temp->coe = c;
    temp->exp = e;
    temp->link = NULL;
    rear->link = temp;
    rear = temp; //rear points to newly
                 inserted last node
    return rear;
}
```



Function to display the Polynomial list

Complete this by modifying the display fn. of SLL

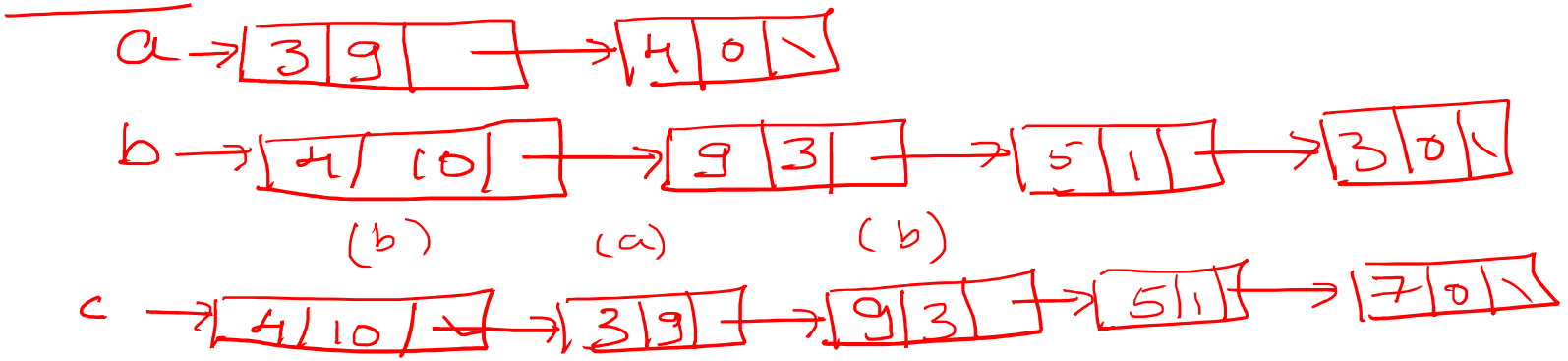
Logic to add two Polynomials represented using list



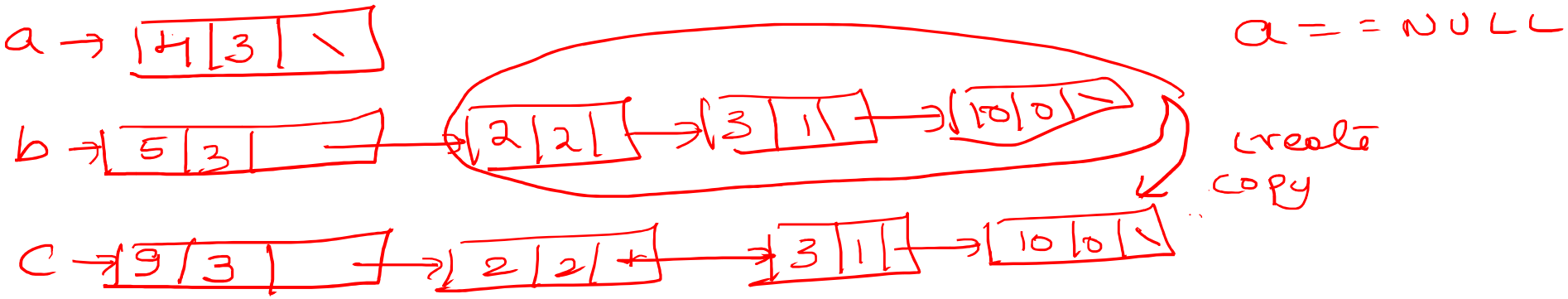

```
if (a->exp == b->exp)
    { rear;
      attach(a->coe + b->coe, a->exp, rear);
        a = a->link;   b = b->link;
    }
else if (a->exp > b->exp)
    { rear = attach(a->coe, a->exp, rear);
      a = a->link; }
else  // a->exp < b->exp
{ rear =
  attach(b->coe, b->exp,
                 rear);
  b = b->link;
}
```

Logic to add two Polynomials represented using list

case(i) different no. of terms in a x b



case(ii)



Function to add two Polynomials represented using list

```
poly *poly::add(poly *a, poly *b)
{  poly *front = new poly;  // dummy node
   poly *rear = front;
   while((a != NULL) && (b != NULL))
   {    switch(compare(a->exp, b->exp)) // write the compare function
        {
         case  1: // a->exp > b->exp
                  rear = reqn(a->coe, a->exp, rear);
                  a = a->link; break;
         case  0: rear = attach(a->coe + b->coe, a->exp, rear);
                  a = a->link; b = b->link; break;
         case -1: rear = attach(b->coe, b->exp, rear)
                  b = b->link;
   } }
```

```
if m > n)
    return 1;
else if m == n)
    return 0;
else return -1;
```

Function to add two Polynomials represented using list

```
for( ; a != NULL; a = a->link)
        rear = attach(a->coe, a->exp, rear);
for( ; b != NULL; b = b->link)
        rear = attach(b->coe, b->exp, rear);

poly *first = front->link;
    delete(front);
return(first);
}
```

To copy remaining nodes to C

main() function

```
int main()
{  poly a, b, c, *a1, *b1,*c1;
 //include appropriate cout statements
    a1= a. polyread();//read polynomial1
    b1=b.polyread(); //read polynomial2
    c1=c.polyadd(a1,b1);
   c.display(c1);
}
```

Polynomial multiplication

- a $3x^2 + 2x + 1$
  b $4x^3 + 2x$

$c = 12x^5 + 6x^3 + 8x^4 + 4x^2 + 4x^3 + 2x$ → Step 1

Sum the terms with same exp. value.

$= 12x^5 + 10x^3 + 8x^4 + 4x^2 + 2x$ ⇒ sort acc. to exp

$= 12x^5 + 8x^4 + 10x^3 + 4x^2 + 2x$

Polynomial multiplication

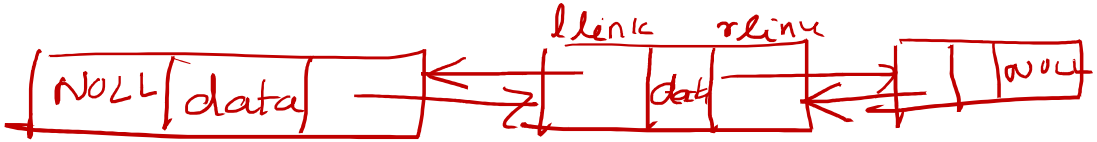
```
poly *poly :: mul(poly *a, poly *b)
{
        ____
        ____
        ____
        while(a!=NULL)
            while(b!=NULL)
                rear = attach(a->coe*b->coe,
                              a->exp+b->exp,
                              rear);
        ____
        first = sum_exp(first); // sum the nodes with same exp
}                                  coe of the
```

Doubly Linked lists

Node structure

Only one node

first node



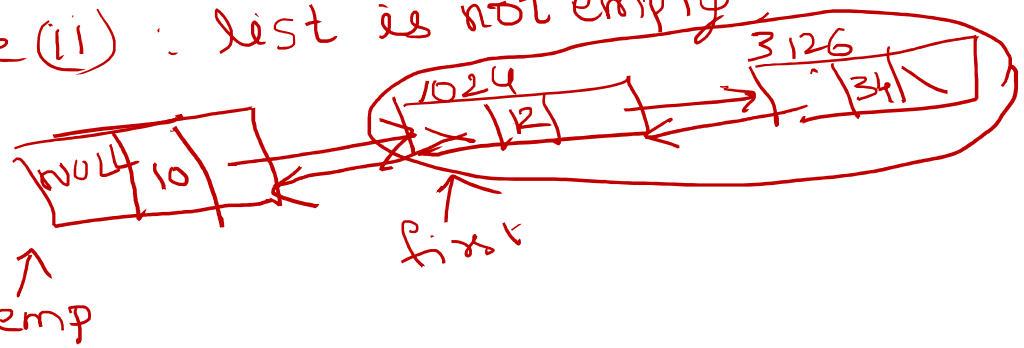
```
class dnode
{
 int info;
 dnode *next;
 dnode *prev;
public:
 dnode* ins_beg(dnode*);
 dnode* ins_end(dnode*);
 void deledata(int);
 void print(dnode*);
};
```

To insert at the beginning

- case (i) : list is empty

```
temp = new dnode;
temp -> data = 10
temp -> prev = temp -> next = NULL;
first = temp;
```

case (ii) : list is not empty;



Same

```
temp -> next = first;
temp -> prev = NULL;
first -> prev = temp;
first = temp;
```

```
dnode* dnode::ins_beg(dnode *head)
{
 dnode *temp=new dnode;
 cout<<"\nInfo: ";
 cin>>temp->info;
 temp->prev=temp->next=NULL;
 if(head==NULL)
 {
   head=temp;
   return head;
 }
 head->prev=temp;
 temp->next=head;
 head=temp;
 return head;
}
```

To insert a node at the end

-

```
dnode *dnode::ins_end(dnode *head)
{
 dnode *temp=new dnode;
 cout<<"\nInfo: ";
 cin>>temp->info;
 temp->prev=temp->next=NULL;
 if(head==NULL)
 {
   head=temp;
   return head;
 }
 dnode *cur=head;
 while(cur->next!=NULL)
 {
  cur=cur->next;
 }
 cur->next=temp;
 temp->prev=cur;
 return head;
```

```
void deldata(int num)
{
   if(head != NULL)
   {
      link * curr, *prev_node, *del_ptr;
      cur_ptr = head;
      prev_ptr = cur_ptr;
      while(cur_ptr->next != NULL)
      {
         if(head->data == num)
         {
            del_ptr = cur_ptr;
            head = cur_ptr->next;
            head->prev = NULL;
            free(del_ptr);
          }
```

```
if(cur_ptr->data == num)
        {
           del_ptr = cur_ptr;
           prev_ptr->next = cur_ptr->next;
           cur_ptr->next->prev = prev_ptr;
           free(del_ptr);
           cur_ptr = prev_ptr;
        }
        prev_ptr = cur_ptr;
        cur_ptr = cur_ptr->next;
    }
```

```
void dnode::print(dnode *head)
{
 dnode *f=head;
 while(f!=NULL)
 {
    cout<<f->info<<"->";
    f=f->next;
}}
```

```
void main()
{
 clrscr();
 dnode d,*head=NULL;
 int c,ele;
 for(;;)
 {
  cout<<"1.Ins b \n 2.Ins e\n 3.Print \n 4.del f\n 5. Del e\n";
  cin>>c;
  switch(c)
  {
   case 1:head=d.insb(head);
                          break;
   case 2:head=d.inse(head);
                          break;
   case 3: d.print(head); break;
   case 4: cout<<“enter element to delete”;
                        cin>>ele;
                        d.deldata(ele);
   default:exit(0);
  }
 }
}
```