**Topic – Polynomial Representation Using Circular linked list and Avail- Singly Linked List**

* **Problem Statement**

Write a menu driven program to do the following:

1. Create a polynomial expression of the form Ax^p +  Bx^q  - C x^r ............ Let it be the current polynomial.  
2. Add another polynomial to the current polynomial  
3. Multiply another polynomial to the current polynomial  
4. Display a polynomial  
5. Delete a polynomial   
6. Delete all linked lists in the memory and exit   
Note: with avail list maintenance Deletion (option 5) should take only constant time.

**Input and Output example:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the details of the polynomial to be created

Enter the number of terms

4

Enter the exponent of the variable x of the 1th term

3

Enter the coefficient of 1th term

5

Enter the exponent of the variable x of the 2th term

2

Enter the coefficient of 2th term

4

Enter the exponent of the variable x of the 3th term

1

Enter the coefficient of 3th term

6

Enter the exponent of the variable x of the 4th term

0

Enter the coefficient of 4th term

2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The current polynomial is P(x) = 5.00x^3 + 4.00x^2 + 6.00x^1 + 2.00

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

2

Enter the details of the polynomial to be added to the current polynomial

Enter the number of terms

3

Enter the exponent of the variable x of the 1th term

4

Enter the coefficient of 1th term

12

Enter the exponent of the variable x of the 2th term

1

Enter the coefficient of 2th term

4

Enter the exponent of the variable x of the 3th term

0

Enter the coefficient of 3th term

15

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The current polynomial is P(x) = 12.00x^4 + 5.00x^3 + 4.00x^2 + 10.00x^1 + 17.00

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

3

Enter the details of the polynomial to be multiplied to the current polynomial

Enter the number of terms

2

Enter the exponent of the variable x of the 1th term

1

Enter the coefficient of 1th term

4

Enter the exponent of the variable x of the 2th term

0

Enter the coefficient of 2th term

3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The current polynomial is P(x) = 48.00x^5 + 56.00x^4 + 31.00x^3 + 52.00x^2 + 98.00x^1 + 51.00

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

There is no polynomial present

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

1

Enter the details of the polynomial to be created

Enter the number of terms

3

Enter the exponent of the variable x of the 1th term

4

Enter the coefficient of 1th term

7

Enter the exponent of the variable x of the 2th term

2

Enter the coefficient of 2th term

6

Enter the exponent of the variable x of the 3th term

1

Enter the coefficient of 3th term

45

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

4

The current polynomial is P(x) = 7.00x^4 + 6.00x^2 + 45.00x^1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1:Create a polynomial

2:Add another polynomial to the current polynomial

3:Multiply another polynomial to the current polynomial

4:Display a polynomial

5:Delete a polynomial

6:Delete all linked lists in the memory and exit

\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*

6

Avail list is already empty

/\* Here the program terminates \*/

* **Proposed C Code**

**/\* ---------- polynomial\_linkedlist.c--------------- \*/**

**#include<stdio.h>**

**#include<stdlib.h>**

**/\* A self referential structure is declared as name Node using typedef keward \*/**

**/\* It contains data and ponter next of same type of Node\*/**

**typedef struct node**

**{**

**int expo;**

**float coefficient;**

**struct node \* next;**

**}Node;**

**/\* Initially avail list is NULL \*/**

**Node\* avail = NULL;**

**/\* function to find the length of the avail list \*/**

**int linkedlistSize(Node\* avail)**

**{**

**Node\* temp = avail;**

**int length = 0;**

**while ( temp != NULL)**

**{**

**temp = temp->next;**

**length++;**

**}**

**temp = NULL;**

**return length;**

**}**

**/\* function to create a node \*/**

**Node\* create(int expo,float coefficient)**

**{**

**Node\* create = (Node\*)malloc(sizeof(Node));**

**create->expo = expo;**

**create->coefficient = coefficient;**

**create->next = NULL;**

**return create;**

**}**

**/\* function to insert a node in a circular linked list in sorted way \*/**

**Node\* circularinsert(Node\* head,int expo,float coefficient)**

**{**

**Node\* temp = create(expo,coefficient);/\* cretaing node \*/**

**if ( head == NULL )/\* if no circular linked list is there so temp will be new head \*/**

**{**

**head = temp;**

**head->next = head ;**

**return head;**

**}**

**else if ( head->expo < expo )/\* inserting the node before head \*/**

**{**

**/\* copying the contents of head to temp \*/**

**temp->expo = head->expo;**

**temp->coefficient = head->coefficient;**

**/\* coping the contents of created node i.e temp to head as a result a swaping occurs \*/**

**head->expo = expo;**

**head->coefficient = coefficient;**

**/\* after swaping temp contains the contents of previous head so it is joined to head->next and head is joined to tem\*/**

**temp->next = head->next;**

**head->next = temp;**

**return head;**

**}**

**else if ( head->next == head )**

**{**

**/\* if only one node is there \*/**

**if ( head->expo == expo) /\* adding when same exponent \*/**

**{**

**head->coefficient = head->coefficient + coefficient;**

**}**

**else**

**{**

**/\* joining the node to last i.e when exponent < head->expo as for lesser and equal conditions are checked \*/**

**temp->next = head->next;**

**head->next = temp;**

**}**

**return head;**

**}**

**else if ( head->next != head )**

**{**

**if ( head->expo == expo )/\* adding for same exponent \*/**

**{**

**head->coefficient = head->coefficient + coefficient;**

**}**

**else**

**{**

**Node\* ptr = head;/\* traversing and checking upto last node \*/**

**while (( ptr->next != head) && ( ptr->next->expo >= expo))**

**{**

**ptr = ptr->next;**

**}**

**if ( ptr->expo == expo )/\* adding for ame exponent \*/**

**{**

**ptr->coefficient = ptr->coefficient + coefficient;**

**}**

**else**

**{**

**/\* joining before the exponent where which is less than that of temp \*/**

**temp->next = ptr->next;**

**ptr->next = temp;**

**}**

**}**

**return head;**

**}**

**}**

**/\* Erasing the polynomial by placing it to the avail list \*/**

**Node\* erasePolynomial(Node\* head)**

**{**

**Node\* temp;**

**if ( head != NULL)**

**{**

**temp = head->next;**

**head->next = avail;**

**avail = temp;**

**head = NULL;**

**}**

**return head;**

**}**

**/\* function to create a polynomial \*/**

**Node\* create\_polynomial(Node\* head)**

**{**

**head = erasePolynomial(head);/\* erasing the previous polynomial if there is any \*/**

**int term,expo;**

**float coefficient;**

**printf("\nEnter the number of terms\n");**

**scanf("%d",&term);**

**for ( int i = 0 ; i < term ; i++ )**

**{**

**printf("Enter the exponent of the variable x of the %dth term\n",i+1);**

**scanf("%d",&expo);**

**printf("Enter the coefficient of %dth term\n",i+1);**

**scanf("%f",&coefficient);**

**head = circularinsert(head,expo,coefficient);/\* insering the term to circular linked list \*/**

**}**

**return head;**

**}**

**/\* function to display the polynomial \*/**

**void display\_polynomial(Node\* head)**

**{**

**if ( head == NULL)**

**{**

**printf("\nThere is no polynomial present\n");**

**}**

**else**

**{**

**printf("\nThe current polynomial is P(x) = ");**

**int headcheck = 0;/\* for checking if all the coefficients are zero \*/**

**Node\* ptr = head;**

**do{**

**if ( ptr->expo == 0)**

**{**

**/\* for zero exponent we will only print the coefficient if it is nonzero \*/**

**if ( ptr->coefficient != 0 )**

**{**

**printf("%0.2f",ptr->coefficient);**

**headcheck = 1;/\* no chance for polynomial to zero so headcheck is 1 \*/**

**}**

**}**

**else if ( ptr->coefficient != 0 )**

**{**

**printf("%0.2fx^%d",ptr->coefficient,ptr->expo);/\* for nonzero coefficient headcheck = 1\*/**

**headcheck = 1;**

**}**

**ptr = ptr->next;**

**/\* we are checking the next node's coefficient upto the last node if it is nonzero then we print '+' \*/**

**if (( ptr != head ) && ( ptr->coefficient != 0))**

**{**

**/\* if the first coefficient is nonzero or whole polynomial is nonzero then we print '+' \*/**

**if( headcheck == 1 )**

**{**

**printf(" + ");**

**}**

**}**

**}while ( ptr != head );/\* checking upto last node \*/**

**if ( headcheck == 0 )/\* headcheck == 0 means the whole polynomial is zero \*/**

**{**

**printf("0.00");**

**}**

**}**

**}**

**/\* deleting a single linear node from the first \*/**

**Node\* delete\_linearnode(Node\* avail)**

**{**

**if ( avail == NULL )**

**{**

**return avail;**

**}**

**else**

**{**

**/\* deleting from the first \*/**

**Node\* ptr = avail;**

**avail = avail->next;**

**free(ptr);**

**}**

**return avail;**

**}**

**/\* deleting the entire avail list \*/**

**Node\* delete\_avail\_list(Node\* avail)**

**{**

**int length = linkedlistSize(avail);**

**if ( length == 0 )**

**{**

**printf("\nAvail list is already empty\n");**

**}**

**for( int i = 0 ; i < length ; i++)**

**{**

**/\* deleting evary node of the avail list \*/**

**avail = delete\_linearnode(avail);**

**}**

**return avail;**

**}**

**/\* function to ass two polynomials \*/**

**Node\* polynomial\_addition(Node\* head,Node\* variablehead)**

**{**

**Node\* headAdd = NULL;/\* it will store the result \*/**

**Node\* temp1 = head;**

**Node\* temp2 = variablehead;/\* it is for the polynomial to be added \*/**

**if ( head == NULL )**

**{**

**head = variablehead;**

**return head;**

**}**

**else**

**{**

**/\* using a do while loop we traverse and add each term to current polynomial and insert in the list \*/**

**/\* if two exponents are same then they will be added in the head by circularinsert function \*/**

**do**

**{**

**head = circularinsert(head,temp2->expo,temp2->coefficient);**

**temp2 = temp2->next;**

**}while ( temp2 != variablehead );**

**headAdd = head;/\* headAdd contains the result \*/**

**head = NULL;**

**}**

**/\* erasing the previous 2 polynomials and storing only the result in head \*/**

**head = erasePolynomial(head);**

**variablehead = erasePolynomial(variablehead);**

**head = headAdd;**

**return head;**

**}**

**/\* function to multiply two polynomials \*/**

**Node\* polynomial\_multiplication( Node\* head,Node\* variablehead)**

**{**

**Node\* multiplyAdd = NULL;/\* it will store the result \*/**

**Node\* temp1 = head;**

**Node\* temp2 = variablehead;/\* it is for the polynomial to be multiplied \*/**

**int expo;**

**float coefficient;**

**if ( head == NULL )**

**{**

**printf("\nThere is no current polynomial so resultant polynomial is P(x) = 0\n");**

**return head;**

**}**

**else**

**{**

**/\* using a nested do while loop we traverse and multiply each term to each term and insert in the list \*/**

**/\* if two exponents are same then they will be added in the multiplyAdd circularinsert function \*/**

**do**

**{**

**do**

**{**

**expo = temp1->expo + temp2->expo;**

**coefficient = temp1->coefficient \* temp2->coefficient;**

**multiplyAdd = circularinsert(multiplyAdd,expo,coefficient);**

**temp2 = temp2->next;**

**}while ( temp2 != variablehead );**

**temp1 = temp1->next;**

**}while ( temp1 != head );**

**}**

**/\* erasing the previous 2 polynomials and storing only the result in head \*/**

**head = erasePolynomial(head);**

**variablehead = erasePolynomial(variablehead);**

**head = multiplyAdd;**

**return head;**

**}**

**int main()**

**{**

**int option;**

**Node\* head = NULL;/\* for the current polynomial \*/**

**Node\* variablehead = NULL;/\* for the polynomial to be added or multiplied \*/**

**do**

**{**

**/\* The menu of options are given \*/**

**printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*Menu of options\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");**

**printf("1:Create a polynomial\n");**

**printf("2:Add another polynomial to the current polynomial\n");**

**printf("3:Multiply another polynomial to the current polynomial\n");**

**printf("4:Display a polynomial\n");**

**printf("5:Delete a polynomial\n");**

**printf("6:Delete all linked lists in the memory and exit\n");**

**printf("\n\*\*\*\*\*\*\*\*Enter option\*\*\*\*\*\*\*\*\n");**

**scanf("%d",&option);**

**switch(option)/\* the cases are placed according to the opertions listed above \*/**

**{**

**case 1:**

**printf("Enter the details of the polynomial to be created\n");**

**head = create\_polynomial(head);**

**break;**

**case 2:**

**printf("Enter the details of the polynomial to be added to the current polynomial\n");**

**variablehead = create\_polynomial(variablehead);**

**/\* after addition the current polynomial is the new head \*/**

**head = polynomial\_addition(head,variablehead);**

**break;**

**case 3:**

**printf("Enter the details of the polynomial to be multiplied to the current polynomial\n");**

**variablehead = create\_polynomial(variablehead);**

**/\* after multiplication the current polynomial is the new head \*/**

**head = polynomial\_multiplication(head,variablehead);**

**break;**

**case 4:**

**display\_polynomial(head);**

**break;**

**case 5:**

**head = erasePolynomial(head);**

**break;**

**case 6:**

**/\* erasing the polynomial and then deleting the entire avail list to free the whole used memory \*/**

**head = erasePolynomial(head);**

**avail = delete\_avail\_list(avail);**

**break;**

**default:**

**printf("Please give option withen (1-6)\n");**

**}**

**}while ( option != 6 );**

**return 0;**

**}**

**/\*------------------------------------------------------------------------------------------------------------------------- \*/**

* **Conclusion**

**The proposed algorithm of the functions like erasing the polynomial by inserting into the avail list takes time of O(1) and displaying the polynomial list or polynomial addition or deleting the enire avail list from memory takes time of O(n) and polynomial multiplication takes time of O(n^2) where n is the length of the entire avail linked list i.e the total nodes used in the program.**

* **Limitations : If the exponents of the polynomial is not integer or negative integer then it will yield wrong output in the result.**
* **Assumptions: Initially the avail linked list is taken as empty. The polynomial should be nonzero and must have exponents as whole numbers.**