**Polynomial Implementation**

#include <stdio.h>

#include <stdlib.h>

struct node{

  int coeff;

  int expo;

  struct node \* next;

};

char filename1[] = "input1.txt";

char filename2[] = "input2.txt";

struct node \* addafter(struct node \*,struct node \*);

struct node \* addatbeg(struct node \*,struct node \*);

struct node \* addatend(struct node \*,struct node \*);

struct node \* create(struct node \*);

struct node \* add(struct node \*,struct node \*);

struct node \* multiply(struct node \*, struct node \*);

struct node \* polEdit(struct node \* );

void displayLL(struct node \*);

struct node \* deleteTerm(struct node \*, int);

struct node \* createFromFile (struct node \*, int fileNumber);

int main(){

  struct node \* polynomial1 = NULL, \* polynomial2 = NULL;

  struct node \* p3 = NULL, \*p4 = NULL;

  int c,status = 0,e;

  printf("---------Polynomial handler program-----------\n");

  do{

    printf("Enter\n 1 to accept 2 polynomials\n ");

    printf("2 to add the 2 polynomials\n ");

    printf("3 to multiply the 2 polynomials\n ");

    printf("4 to modify either of two polynomials\n ");

    printf("5 to display both polynomials\n ");

    printf("6 to accept two polynomials from input file:\n ");

    printf("7 to exit the program\n ");

    scanf("%d",&c);

    switch(c){

      case 1:

        printf("Enter polynomial p1 :\n");

        polynomial1 = create(polynomial1);

        printf("Enter polynomial p2 :\n");

        polynomial2 = create(polynomial2);

        status = 1;

        break;

      case 2:

        if (status == 0){

          printf("Polynomials have to be accepted\n");

          polynomial1 = create(polynomial1);

          polynomial2 = create(polynomial2);

          status = 1;

        }

        p3 = add(polynomial1, polynomial2);

        displayLL(p3);

        break;

      case 3:

        if (status == 0){

          printf("Polynomials have to be accepted\n");

          polynomial1 = create(polynomial1);

          polynomial2 = create(polynomial2);

          status = 1;

        }

        p4 = multiply(polynomial1,polynomial2);

        displayLL(p4);

        break;

      case 4:

        printf("Which polynomial to you wish to modify(1-p1,2-p2):");

        scanf("%d", &e);

        if (e==1){

          polynomial1 = polEdit(polynomial1);

        }else if(e==2){

          polynomial2 = polEdit(polynomial2);

        }

        break;

      case 5:

        displayLL(polynomial1);

        displayLL(polynomial2);

        break;

      case 6:

        polynomial1 = createFromFile(polynomial1,1);

        printf("Input from input1.txt:\n");

        displayLL(polynomial1);

        polynomial2 = createFromFile(polynomial2,2);

        printf("Input from input2.txt:\n");

        displayLL(polynomial2);

        status = 1;

        break;

      case 7:

        break;

      default:

        printf("Enter correct input values\n");

    }

  }while(c!=7);

  return 0;

}

struct node \* create(struct node \* start){

  //start == NULL

  start = NULL;

  struct node \* temp;

  int n,a,b;

  printf("Enter number of terms:");

  scanf("%d",&n);

  while (n--){

    temp = (struct node \*)malloc(sizeof(struct node));

    printf("Enter coefficient and exponent (c,e):");

    scanf("%d,%d", &(temp->coeff),&(temp->expo));

    temp->next = NULL;

    if (start == NULL){

      start = addatbeg(start,temp);

    }else{

      start = addafter(start, temp);

    }

  }

  return start;

}

struct node \* addatbeg(struct node \* start, struct node \* p){

  p->next = start;

  start = p;

  return start;

}

struct node \* addatend(struct node \* start, struct node \* p){

    struct node \* temp = start;

    if (temp==NULL){

      start = p;

      return start;

    }

    while (temp->next != NULL )

      temp = temp->next;

    temp->next = p;

    return start;

}

struct node \* addafter(struct node \* start, struct node \* p){

  //check if p's coefficient < the next node's coefficient

  //check first node is null or not

  if ((start==NULL) || (p->expo > start->expo)){

    start = addatbeg(start,p);

    return start;

  }

  struct node \* temp = start;

  while ((temp->next != NULL) && (p->expo < temp->next->expo)){

    temp = temp->next;

  }

  //check if temp->next = null, if it is, add at end

  if (temp->next == NULL){

    //check whether one node only

    if (p->expo > temp->expo){

      p->next = temp;

      start = p;

    }

    else if (p->expo == temp->expo){

      temp->coeff += p->coeff;

    }else if (p->expo < temp->expo){

      temp->next = p; //if the p exp is lower than any yet encountered

    }

    return start;

  }else {

    if (p->expo == temp->next->expo)

      temp->next->coeff += p->coeff;

    else if (p->expo > temp->next->expo) {

      p->next = temp->next;

      temp->next = p;

    }

  }

  return start;

}

void displayLL(struct node \* a){

    if (a==NULL){

        printf("Empty list\n");

        return;

    }

    //int i = 0;

    printf("| ");

    while (a!= NULL){

        printf("%dX^%d + ",a->coeff, a->expo);

        a = a->next;

        //i++;

    }

    printf(" + 0 = 0 |\n");

}

struct node \* add(struct node \*a,struct node \*b){

  //add two polynomials

  struct node \* sum = NULL;

  while (a!=NULL && b!= NULL)//as long as neither one  null

  {

    struct node \* temp = (struct node \*)malloc(sizeof(struct node));

    if (a->expo == b->expo){

      temp->expo = b->expo;

      temp->coeff = a->coeff + b->coeff;

      temp->next = NULL; //not needed

      sum = addafter(sum,temp);

      a = a->next;

      b = b->next;

    }else if (a->expo > b->expo){

      temp->expo = a->expo;

      temp->coeff = a->coeff;

      temp->next = NULL;

      sum = addafter(sum,temp);

      a = a->next;

    }else if (b->expo > a->expo){

      temp->expo = b->expo;

      temp->coeff = b->coeff;

      temp->next = NULL;

      sum = addafter(sum,temp);

      b = b->next;

    }

  }

  struct node \* p;

  //handle the other one

  p = (a==NULL)?b:a;

  while (p!=NULL){

    struct node \* temp = (struct node \*)malloc(sizeof(struct node));

    temp->coeff = p->coeff;

    temp->expo = p->expo;

    temp->next = NULL;

    sum = addafter(sum, temp);

    p = p->next;

  }

  return sum;

}

struct node \* multiply(struct node \* p1, struct node \*p2){

  //multiplication valid as long as both are non null

  struct node \* product = NULL;

  struct node \* a, \*b;

  a = p1;

  b = p2;

  if (a==NULL || b==NULL)

    return product;

  /\*

  carry out outer traversal on one polynomial

  inner traversal occurs as long as outer is not null

  use just one product node

  \*/

 while (a!=NULL){

  b = p2;

  while (b!=NULL){

    struct node \* temp = (struct node \*)malloc(sizeof(struct node));

    temp->coeff = a->coeff \* b->coeff;

    temp->expo = a->expo + b->expo;

    temp->next = NULL;

    product = addafter(product, temp);

    b = b->next;

  }

  a = a->next;

 }

 return product;

}

struct node \* polEdit(struct node \* a){

  //either insert a term or delete a term

  int c;

  printf("Enter \n\t1 to insert a term\n\t2 to delete a term\n:");

  scanf("%d", &c);

  struct node \* temp = (struct node \*)malloc(sizeof(struct node));

  if (c==1)

  {

    printf("Enter coefficient and exponent:");

    scanf("%d,%d",&(temp->coeff),&(temp->expo));

    temp->next = NULL;

    a = addafter(a,temp);

  }

  else if (c==2){

    printf("Enter exponent of term to be deleted:");

    scanf("%d",&c);

    a = deleteTerm(a, c);

  }

  else{

    printf("Only two options\nReturning unmodified start\n");

  }

  return a;

}

struct node \* deleteTerm(struct node \* start, int exp){

  struct node \* p = start;

  if (start == NULL)

    return start;

  else if (start->expo == exp){

    p = start;

    start = start->next;

    free(p);

    return start;

  }

  while (p->next!=NULL){

    if (p->next->expo == exp){

      struct node \* temp = (struct node \*)malloc(sizeof(struct node));

      temp = p->next;

      p->next = temp->next;

      free(temp);

      return start;

    }

    p = p->next;

  }

  printf("Item %d not found in list\n");

  return start;

}

struct node \* createFromFile (struct node \* start, int n){

  start = NULL;

  FILE \* fp1 = fopen(filename1,"r");

  FILE \* fp2 = fopen(filename2,"r");

  FILE \* fp;

  if (n==1)

    fp = fp1;

  else if (n==2)

    fp = fp2;

  //first line of every input file contains number of terms

  //have to take that many terms

  // an example term is (2,3) in coefficient,exponent format

  //each term is space seperated

  int n1,n2,c,e,y;

  fscanf(fp,"%d",&n1);

  struct node \* temp = NULL;

  for (y = 0 ; y < n1; y++){

    fscanf(fp,"%d %d",&c,&e);

    temp = (struct node \*)malloc(sizeof(struct node));

    temp->coeff = c;

    temp->expo = e;

    temp->next = NULL;

    start = addafter(start,temp);

  }

  return start;

}

**Sets Implementation**

#include <stdio.h>

#include <stdlib.h>

struct node{

  int value;

  struct node \* next;

};

struct node \* addafter(struct node \*,struct node \*);

struct node \* addatbeg(struct node \*,struct node \*);

struct node \* addatend(struct node \*,struct node \*);

struct node \* create(struct node \*);

struct node \* setEdit(struct node \* );

void displayLL(struct node \*);

struct node \* deleteTerm(struct node \*, int);

int isIn(struct node \*, int);

struct node \* Set\_union(struct node \*,struct node \*);

struct node \* difference(struct node \*,struct node \*);

struct node \* intersection(struct node \*,struct node \*);

int main(){

  struct node \* set1 = NULL, \* set2 = NULL, \* set3 = NULL, \*set4 = NULL;

  int c,status = 0,e;

  printf("---------Set handler program-----------\n");

  do{

    printf("Enter\n 1 to accept 2 sets\n ");

    printf("2 to perform set union\n ");

    printf("3 to perform set intersection\n ");

    printf("4 to perform set difference\n ");

    printf("5 to display both sets\n ");

    printf("6 to exit \n ");

    scanf("%d",&c);

    switch(c){

      case 1:

        printf("Enter set A :\n");

        set1 = create(set1);

        printf("Enter set p2 :\n");

        set2 = create(set2);

        status = 1;

        break;

      case 2:

        if (status == 0){

          printf("Sets have to be accepted before union\nEnter Set A:\n");

          set1 = create(set1);

          printf("Enter set B:\n");

          set2 = create(set2);

          status = 1;

        }

        set3 = Set\_union(set1, set2);

        displayLL(set3);

        break;

      case 3:

      if (status == 0){

        printf("Sets have to be accepted before intersection\nEnter Set A:\n");

        set1 = create(set1);

        printf("Enter set B:\n");

        set2 = create(set2);

        status = 1;

      }

        set4 = intersection(set1,set2);

        displayLL(set4);

        break;

      case 4:

      if (status == 0){

        printf("Sets have to be accepted before difference\nEnter Set A:\n");

        set1 = create(set1);

        printf("Enter set B:\n");

        set2 = create(set2);

        status = 1;

      }

        printf("Enter 1 for A-B, 2 for B-A:");

        scanf("%d", &e);

        if (e==1){

          set3 = difference(set1,set2);

        }else if(e==2){

          set3 = difference(set2,set1);

        }else{

          printf("Invalid order\n");

        }

        displayLL(set3);

        break;

      case 5:

        printf("Set 1: ");

        displayLL(set1);

        printf("Set 2: ");

        displayLL(set2);

        break;

      case 6:

        break;

      default:

        printf("Enter correct input values\n");

    }

  }while(c!=6);

  return 0;

}

struct node \* create(struct node \* start){

  //start == NULL

  start = NULL;

  struct node \* temp;

  int n,a,b;

  printf("Enter number of items:");

  scanf("%d",&n);

  while (n--){

    temp = (struct node \*)malloc(sizeof(struct node));

    printf("Enter value:");

    scanf("%d", &(temp->value));

    temp->next = NULL;

    if (start == NULL){

      start = addatbeg(start,temp);

    }else{

      start = addafter(start, temp);

    }

  }

  return start;

}

struct node \* addatbeg(struct node \* start, struct node \* p){

  p->next = start;

  start = p;

  return start;

}

struct node \* addatend(struct node \* start, struct node \* p){

    struct node \* temp = start;

    if (temp==NULL){

      start = p;

      return start;

    }

    while (temp->next != NULL )

      temp = temp->next;

    temp->next = p;

    return start;

}

struct node \* addafter(struct node \* start, struct node \* p){

  if ((start==NULL) || (p->value < start->value)){

    start = addatbeg(start,p);

    return start;

  }

  struct node \* temp = start;

  while ((temp->next != NULL) && (p->value > temp->next->value)){

    temp = temp->next;

  }

  //check if temp->next = null, if it is, add at end

  if (temp->next == NULL){

    if (p->value > temp->value){

      temp->next = p;

    }

  }else {

    if (p->value < temp->next->value) {

      p->next = temp->next;

      temp->next = p;

    }

  }

  return start;

}

void displayLL(struct node \* a){

    if (a==NULL){

        printf("Empty list\n");

        return;

    }

    printf("{ ");

    while (a!= NULL){

        printf("%d , ", a->value);

        a = a->next;

        //i++;

    }

    printf(" }\n");

}

struct node \* Set\_union(struct node \*a,struct node \*b){

  //add two sets

  struct node \*sum = NULL;

  while (a!=NULL && b!= NULL)//as long as neither one  null

  {

    struct node \* temp = (struct node \*)malloc(sizeof(struct node));

    if (a->value == b->value){

      temp->value = a->value;

      temp->next = NULL;

      if (sum == NULL){

        sum = addatbeg(sum,temp);

      }

      else {

        sum = addatend(sum,temp);

      }

      a = a->next;

      b = b->next;

    }else if (a->value < b->value){

      temp->value = a->value;

      temp->next = NULL;

      sum = addatend(sum,temp);

      a = a->next;

    }else if (b->value < a->value){

      temp->value = b->value;

      temp->next = NULL;

      sum = addatend(sum,temp);

      b = b->next;

    }

  }

  struct node \* p;

  //handle the other one

  p = (a==NULL)?b:a;

  while (p!=NULL){

    struct node \* temp = (struct node \*)malloc(sizeof(struct node));

    temp->value = p->value;

    temp->next = NULL;

    sum = addatend(sum, temp);

    p = p->next;

  }

  return sum;

}

int isIn(struct node \* a, int v){

  int s = 0;

  while (a!= NULL){

    if (a->value == v){

      return 1;

    }

    a = a->next;

  }

  return 0;

}

struct node \* intersection(struct node \*a,struct node \*b){

  if (a==NULL || b==NULL)

    return NULL;

  struct node \* inter = NULL;

  int i;

  while (a && b){

    i = a->value;

    if (isIn(b,i)){

      struct node \* temp = (struct node \* )malloc(sizeof(struct node));

      temp->value = i;

      temp->next = NULL;

      inter = addatend(inter,temp);

    }

    a = a->next;

  }

  return inter;

}

struct node \* difference(struct node \* from, struct node \* remove){

  struct node \* inter = intersection(from, remove);

  //iterate over from,

  //if element in inter, don't input

  struct node \* result = NULL;

  for (from; from != NULL; from = from->next){

    if (!(isIn(inter,from->value))){

      struct node \* temp = (struct node \*)malloc(sizeof(struct node));

      temp->value = from->value;

      temp->next = NULL;

      result = addatend(result,temp);

    }

  }

  return result;

  //as we have to remove from `from` all inter elements

}

struct node \* deleteTerm(struct node \*start, int v){

  struct node \* a = start;

  if (a == NULL)

    return a;

  else if (!(isIn(a,v)))

    return a;

  //start element

  else if (a->value == v){

    struct node \* temp = a;

    a = a->next;

    free(temp);

    return a;

  }else {

    while (a->next != NULL){

      if (a->next->value == v){

        struct node \* temp = a->next;

        a->next = temp->next;

        free(temp);

        return a;

      }

      a = a->next;

    }

  }

}