Institute of Engineering & Management
Department of Computer Science & Engineering
Data Structure Laboratory for 2nd year 3rd semester 2017
Code: CS 392

Date: 30/8/17

ASSIGNMENT-4(Continued)

Problem-2

Problem Statement: Adding two polynomial expressions using Linked List

Algorithm:

Step-1: START
Step-2: define a type NODE of structure containing coeff & expo as integer and a NODE pointer next
Step-3: declare glabally head1, head2, head3 as NODE pointer
Step-4: define a function alloc() & return (NODE *)malloc(sizeof(NODE))
Step-5: Inside main(), declare 2 charater arrays str1[100], str2[100]
Step-6: print the command and scan for string for 1st polynomial in str1
Step-7: print the command and scan for string for 2nd polynomial in str2
Step-8: head1 = str_to_ll(str1, head1)
Step-9: head2 = str_to_ll(str2, head2)
Step-10: call addition() & free all the nodes using a loop & temp pointer
Step-11: Inside str_to_ll(char arr[], NODE *head), declare integer variables i=0, flag_ex=0, neg=0, total=0, flag=0, count=0 and NODE pointer temp & new
Step-12: do (repeat)

```
if arr[i] = ' - ' then neg = neg + 1
if arr[i] >= '0' and arr[i] <='9'
        total = (total*10) + arr[i] - '0'
        flag = flag + 1
else if (arr[i] <='0' or arr[i]>='9') and flag!=0
        if count = 0
                new = alloc()
        if new = NULL
                print error and exit
        if flag_ex != 0 then
                if neg = 0 then new -> expo = 0 – total
                else new -> expo = total & neg = 0
        else if neg = 0 then new -> coeff = 0 – total
                else new -> coeff = total & neg = 0
flag = flag_ex = total = 0 and count = count + 1
if arr[i] = '^'
        flag_ex = flag_ex + 1
if count = 2 then
        if head = NULL then temp = head = new
        temp -> next = new
        temp = temp -> next & count = 0
```

while arr[i] != '\0' & i = i + 1

Step-13: temp -> next = NULL & return head
Step-14: Inside addition(), declare NODE pointer temp, new & prev
Step-15: while head1 != NULL or head2 != NULL

```
if head1 = NULL
        while head2 != NULL
                new=alloc()
                if new = NULL
                        print “Error(could not allocate
                                memory)” & exit(1)
                new->next = NULL
                new->coeff = head2->coeff
                new->expo = head2->expo
                prev = head2 & head2 = head2->next & free(prev)
                if head3 = NULL then head3=new
                else temp -> next = new
                temp = new
else if head2 = NULL
        while head1 != NULL
                new = alloc()
                if new = NULL
                        print "Error(could not allocate
                                memory)" & exit(1)
                new -> next = NULL
                new -> coeff=head1 -> coeff
                new -> expo = head1 -> expo
                prev = head1 & head1 = head1->next & free(prev)
                if head3 = NULL then head3=new
                else temp -> next = new
                temp = new
else if head1 -> expo = head2 -> expo
        new = alloc()
        if new = NULL
                print "Error(could not allocate memory)" & exit(1)
        new -> next = NULL
        new -> coeff = head1 -> coeff + head2 -> coeff
        new -> expo = head1 -> expo
        prev = head1 & head1 = head1 -> next & free(prev)
        prev = head2 & head2 = head2 -> next & free(prev)
        if head3 = NULL then head3 = new
        else temp -> next = new & temp = new
else if head1 -> expo > head2 -> expo
        new = alloc()
        if new = NULL
        print "Error(could not allocate memory)" & exit(1)
        new -> next = NULL
        new -> coeff = head1 -> coeff
        new -> expo = head1 -> expo
        prev = head1 & head1 = head1 -> next & free(prev)
        if head3 = NULL then head3 = new
        else temp -> next = new & temp=new;
else if head1 -> expo < head2 -> expo
        new = alloc()
        if new = NULL
                print "Error(could not allocate memory)" & exit(1)
```

new -> next = NULL
new -> coeff = head2 -> coeff
new -> expo = head2 -> expo
prev = head2 & head2 = head2 -> next & free(prev)
if head3 = NULL then head3 = new
else temp -> next = new & temp = new

Step-16: Inside display(NODE *temp), print “the final expression is ”

Step-17: while temp != NULL repeat
print “(temp -> coeff)*x^(temp -> expo)”
if temp -> next != NULL
print “+”
temp = temp -> next

Step-18: print “\n”

Step-19: END

Source code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int coeff, expo;
    struct node *next;
} NODE;

NODE *head1=NULL, *head2=NULL, *head3=NULL;

NODE *alloc()
{
    return (NODE *)malloc(sizeof(NODE));
}

NODE *str_to_ll(char *, NODE *);
void addition();
void display(NODE *);

int main()
{
    char str1[100], str2[100];
    printf("Enter the simplified 1st polynomial expression
                (single variable)\n");
    scanf("%[^\n]s", str1); fflush(stdin);
    printf("Enter the simplified 2nd polynomial expression
                (single variable)\n");
    scanf("%[^\n]s", str2);
    head1=str_to_ll(str1, head1);
    head2=str_to_ll(str2, head2);
    addition(); display(head3);
    NODE *temp=head3;
    while(head3!=NULL)
    {
        temp=head3->next; free(head3);
        head3=temp;
    }
    return 0;
}
```

```
NODE *str_to_ll(char arr[], NODE *head)
{
    int i=0, flag_ex=0, neg=0, total=0, flag=0,
           count=0;NODE *temp, *new;
    do
    {
        if(arr[i]=='-')
            neg++;
        if(arr[i]>='0' && arr[i]<='9')
        {
            total=(total*10)+arr[i]-'0';
            flag++;
        }
        else if((arr[i]<'0' || arr[i]>'9') && flag!=0)
        {
            if(count==0)
                new=alloc();
            if(new==NULL)
            {
                printf("Error(could not allocate
                            memory)\n"); exit(1);
            }
            if(flag_ex!=0)
                if(neg==0)
                    new->expo=total;
                else { new->expo=0-total; neg=0; }
            else
                if(neg==0)
                    new->coeff=total;
                else { new->coeff=0-total; neg=0; }
            flag=flag_ex=total=0; count++;
        }
        if(arr[i]=='^')
            flag_ex++;
        if(count==2)
        {
            if(head==NULL)
            {
                head=new; temp=head;
            }
            temp->next=new;
            temp=temp->next;
            count=0;
        }
    }while(arr[i++]!='\0');
    temp->next=NULL;
    return head;
}

void addition()
{
    NODE *temp, *new, *prev;
    while(head1!=NULL || head2!=NULL)
    {
        if(head1==NULL)
            while(head2!=NULL)
            {
                new=alloc();
```

```
            if(new==NULL)
            {
                printf("Error(could not allocate
                        memory)\n"); exit(1);
            }
            new->next=NULL;
            new->coeff=head2->coeff;
            new->expo=head2->expo;
            prev=head2; head2=head2->next; free(prev);
            if(head3==NULL)
                  head3=new;
            else temp->next=new;
            temp=new;
        }
    else if(head2==NULL)
        while(head1!=NULL)
        {
            new=alloc();
            if(new==NULL)
            {
                printf("Error(could not allocate
                              memory)\n"); exit(1);
            }
            new->next=NULL;
            new->coeff=head1->coeff;
            new->expo=head1->expo;
            prev=head1; head1=head1->next; free(prev);
            if(head3==NULL)
                  head3=new;
            else temp->next=new;
            temp=new;
        }
    else if(head1->expo==head2->expo)
    {
        new=alloc();
        if(new==NULL)
        {
            printf("Error(could not allocate
                        memory)\n"); exit(1);
        }
        new->next=NULL;
        new->coeff=head1->coeff+head2->coeff;
        new->expo=head1->expo;
        prev=head1; head1=head1->next; free(prev);
        prev=head2; head2=head2->next; free(prev);
        if(head3==NULL)
            head3=new;
        else temp->next=new;
        temp=new;
    }
    else if(head1->expo>head2->expo)
    {
        new=alloc();
        if(new==NULL)
        {
            printf("Error(could not allocate
                        memory)\n"); exit(1);
        }
```

```
                new->next=NULL;
                new->coeff=head1->coeff;
                new->expo=head1->expo;
                prev=head1; head1=head1->next; free(prev);
                if(head3==NULL)
                    head3=new;
                else temp->next=new;
                temp=new;
            }
            else if(head1->expo<head2->expo)
            {
                new=alloc();
                if(new==NULL)
                {
                    printf("Error(could not allocate
                                memory)\n"); exit(1);
                }
                new->next=NULL;
                new->coeff=head2->coeff;
                new->expo=head2->expo;
                prev=head2; head2=head2->next; free(prev);
                if(head3==NULL)
                    head3=new;
                else temp->next=new;
                temp=new;
            }
        }
    }

    void display(NODE *temp)
    {
        printf("The final expression is\n");
        while(temp!=NULL)
        {
            printf("[(%d)*x^(%d)]", temp->coeff, temp->expo);
            if(temp->next!=NULL)
                printf("+");
            temp=temp->next;
        }
        printf("\n");
    }
```

Input/Output: Enter the simplified 1st polynomial expression (single variable)
8x^3 + 2x^2 + 3x^1 + 5x^0
Enter the simplified 2nd polynomial expression (single variable)
4x^4 - 2x^3 - 2x^1 + 1x^1 + 2x^(-1)
The final expression is
[(4)*x^(4)]+[(6)*x^(3)]+[(2)*x^(2)]+[(1)*x^(1)] +[(5)*x^(0)] +[(1)*x^(-1)]

Problem-3

Problem Statement: Implement Josephus problem using Linked List

Algorithm:

Step-1: START
Step-2: define type NODE of Structure containing integer variable num & NODE pointer next
Step-3: Declare globally first = NULL and last = NULL as NODE pointer
Step-4: Inside main(), declare in as integer variable & temp as NODE pointer
Step-5: print “enter the no. of players”
Step-6: scan for in & if in = 0, then return
Step-7: call create(in) & print “enter the no. of intervals”
Step-8: scan for in & if in = 0, then return
Step-9: call jsph(in) & last = NULL & temp = first
Step-10: use a loop to free the allocated nodes
Step-11: Inside create(int i), declare NODE pointer new & print user command
Step-12: while i > 0 repeat
    if last = NULL, then first = last = new & first -> next = NULL
    last -> next = new & new -> next = first & last = new
    scan for last -> num & i = i + 1
Step-13: Inside jsph(int I), declare integer variable j and & NODE pointer temp1 = first & temp2
Step-14: while temp1 != temp1 -> next, then repeat
    temp1 = temp1 -> next
Step-15: temp2 = temp1 -> next
Step-16: temp1 -< next = temp2 -> next
Step-17: free(temp2) & temp1 = temp1 -> next
Step-18: print “The winner is temp -> num”
Step-19: END

Source code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node

    int num;
    struct node *next;
} NODE;

NODE *first=NULL, *last=NULL;

void create(int);
void jsph(int);

void main()
{
    int in; NODE *temp;
    printf("enter the no of players\n");
    scanf("%d",&in);
    if(in==0)
        return;
    create(in);
    printf("enter the no of intervals\n");
    scanf("%d",&in);
```

```
    if(in==0)
        return;
    jsph(in);
    last=NULL; temp=first;
    while(temp!=NULL)
    {
        temp=first->next;
        free(first);
        first=temp;
    }
}

void create(int i)
{
    NODE *new;
    printf("Enter the player separated by spaces\n");
    while(i>0)
    {
        new=(NODE *)malloc(sizeof(NODE));
        if(last==NULL)
        {
            last=first=new;
            first->next=last;
        }
        last->next=new;
        new->next=first;
        last=new;
        scanf("%d",&last->num);
        i--;
    }
}

void jsph(int i)
{
    NODE *temp1=first, *temp2; int j;
    while(temp1!=temp1->next)
    {
        for(j=i;j>1;j--)
            temp1=temp1->next;
        temp2=temp1->next;
        temp1->next=temp2->next;
        free(temp2);
        temp1=temp1->next;
    }
    printf("the winner is %d\n", temp1->num);
}
```

Input/Output:
enter the no of players
6
Enter the player separated by spaces
1 2 3 4 5 6
enter the no of intervals
1
the winner is 5

Problem-4

Problem Statement: Implement Stack using Linked List

Algorithm:

Step-1: START
Step-2: define a type NODE of structure containing integer variable num and NODE pointer next
Step-3: Declare global NODE pointer top = NULL
Step-4: define a function alloc() & return (NODE *)malloc(sizeof(NODE))
Step-5: Inside main(), declare rpt=1, i=0 as integers and a NODE pointer temp
Step-6: do (repeat)
    Print the commands for user
    Scan for i.
    Switch for values of i between
        case 1: call push()
        case 2: call pop()
        case 3: call display()
        default: print "wrong input".
    Ask user whether to continue or exit
    scan for rpt
  while rpt is equal to 1
Step-7: free all the nodes using a loop to prevent memory leakage
Step-8: inside push(), declare temp as NODE pointer
Step-9: if temp = NULL, then print "error" & return
  else scan for temp -> num
    temp -> next = top & top = temp
Step-10: inside pop(), declare NODE pointer temp
Step-11: if top is equal to NULL
        print "Stack underflow" & return
  else temp = top -> next & free(top)
    top = temp & print "deleted"
Step-12: inside display(),declare NODE pointer temp = top
Step-13: if top is equal to NULL, then print "Stack empty"
  else print every element in the stack from position 0 to top
Step-14: END

Source code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
      int num; struct node *next;
} NODE;

NODE *top=NULL;

NODE *alloc()
{return (NODE *)malloc(sizeof(NODE));}
void pop();
void push();
void display();

void main()
{
```

```
	NODE *temp; int rpt=1, i=0;
	printf("Choose between following operation\n '1' to
		push operation\n '2' for pop operation\n '3' to
				display\n");
	do
	{
		printf("Enter the operation command\n");
		scanf("%d",&i);
		switch(i)
		{
			case 1: push(); break;
			case 2: pop(); break;
			case 3: display(); break;
			default: printf("Wrong input\n");
				continue;
		}
		printf("Do u want to continue? if yes then press
				'1' or else press any key\n");
		fflush(stdin); scanf("%d", &rpt);
	} while (rpt == 1); temp=top;
	while(top!=NULL)
	{
		temp = top; top = top->next; free(temp);
	}
}

void display()
{
	NODE *temp=top;
	if(top==NULL)
	{
		printf("Stack Empty\n"); return;
	}
	printf("The elements in the stack are \n");
	while(temp!=NULL)
	{
		printf("%d, ",temp->num);
		temp=temp->next;
	}
}

void push()
{
	NODE *temp=alloc();
	if(temp==NULL)
	{
		printf("Unable allcate memory\n"); return;
	}
	else
	{
		printf("Enter the integer value\n");
		scanf("%d", &temp->num);
		temp->next=top; top=temp;
	}
}

void pop()
{
```

```
        NODE *temp;
        if(top==NULL)
        {
              printf("Stack underflow\n"); return;
        }
        else
        {
              temp=top->next;
              free(top); top=temp;
              printf("deleted\n");
        }
}
```

Input/Output:

```
Choose between following operation
 '1' to push operation
 '2' for pop operation
 '3' to display
Enter the operation command
1
Enter the integer value
23
Do u want to continue? if yes then press '1' or else press any key
1
Enter the operation command
1
Enter the integer value
45
Do u want to continue? if yes then press '1' or else press any key
1
Enter the operation command
1
Enter the integer value
67
Do u want to continue? if yes then press '1' or else press any key
1
Enter the operation command
3
The elements in the stack are
67, 45, 23, Do u want to continue? if yes then press '1' or else press any key
1
Enter the operation command
2
deleted
Do u want to continue? if yes then press '1' or else press any key
1
Enter the operation command
3
The elements in the stack are
45, 23, Do u want to continue? if yes then press '1' or else press any key
0
```

Problem-5

Problem Statement: Implement Simple Queue using Linked List

Algorithm:

Step-1: START
Step-2: define type NODE as structure containing integer num & NODE pointer next
Step-3: Declare global NODE pointers front=rear=NULL
Step-4: Inside main(), declare flag=1 in as integers and temp as NODE pointer
Step-5: Repeat
    Print the commands for user
    Scan for in.
    Switch for values of i between
        case 1: call insert()
        case 2: call del()
        case 3: call display()
        default: print “wrong input”.
    Ask user whether to continue or exit
    scan for flag
  while flag is equal to 1
Step-6: free all the nodes using a loop to prevent memory leakage
Step-7: Inside create_node(int i),
    declare NODE pointer new=(NODE *)malloc(sizeof(NODE))
Step-8: if new = NULL, then print “error” & call exit(1)
Step-9: if front = NULL, then front = rear = new
  else rear -> next = new & rear = new
Step-10: rear -> num = i & rear -> next = NULL
Step-11: inside insert(), declare variables I, n=0, flag=0, flag1=0, len & character
    array buffer[100]
Step-12: print “enter the data separated by spaces”
Step-13: fflush(stdin) & gets(buffer)
Step-14:len = strlen(buffer)
Step-15: if len = 0, then print “no input” & return
Step-16: for i = 0 to i = len repeat
    if buffer[i] = ‘-’
        flag = flag + 1 & continue
    if buffer[i] = ‘ ‘ and buffer[i] = ‘\0’
        n = (n*10) + (buffer[i] - ‘0’) & continue
    if flag is not equal to 0
        rear = rear+1 & create_node(-n)
    else rear = rear+1 & create_node(n)
    assigne n = 0 & flag = 0
Step-17: inside del(), if rear = NULL, then print “Queue empty” & return
Step-18: declare NODE pointer temp & print “deleted”
Step-19: if rear = front
    free(rear) & rear = front = NULL
  else temp = front & front = front -> next & free(temp)
Step-20: inside display(), if rear = NULL
    print “Queue empty” & return
Step-21: while temp != NULL repeat
    print “temp -> num” & temp = temp -> next
Step-22: END

Source code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct node
{
      int num;
      struct node *next;
} NODE;

NODE *front=NULL, *rear=NULL;

void create_node(int);
void delete();
void insert();
void display();

void main()
{
      NODE *temp;
      int in,flag=0;
      printf("Enter the following commands\n '1' to insert\n
                  '2' to delete\n '3' to display\n");
      do
      {
            printf("Enter the Command\n");
            scanf("%d",&in);
            switch(in)
            {
                  case 1:     insert(); break;
                  case 2:     delete(); break;
                  case 3:     display(); break;
                  default: printf("wrong input\n");
            }
            printf("enter 1 to continue\n");
            scanf("%d",&flag);
      } while(flag==1);
      temp=front;
      while(front!=NULL)
      {
            temp=front;
            front=front->next;
            free(temp);
      }
}

void create_node(int i)
{
      NODE *new=(NODE *)malloc(sizeof(NODE));
      if(new==NULL)
      {
            printf("Could not allocate memory\n");
            exit(1);
      }
      if(front==NULL)
            front=rear=new;
      else{
            rear->next=new;
```

```
            rear=new;
      }
      rear->next=NULL;
      rear->num=i;
}

void insert()
{
      int i, n=0, flag=0, len;
      char buffer[200];
      printf("enter the data separated by spaces\n");
      fflush(stdin); gets(buffer);
      len=strlen(buffer);
      if(len<1)
      {
            printf("no input\n");
            return;
      }
      for(i=0;i<=len;i++)
      {
            if(buffer[i]=='-')
            {
                  flag++; continue;
            }
            if(buffer[i]!=' ' && buffer[i]!='\0')
            {
                  n=(n*10)+(buffer[i]-'0');
                  continue;
            }
            if(flag!=0)
                  create_node(0-n);
            else create_node(n);
            n=0; flag=0;
      }
}

void delete()
{
      NODE *temp;
      if(rear==NULL)
      {
            printf("queue empty\n"); return;
      }
      printf("deleted\n");
      if(rear==front)
      {
            free(front);
            rear=front=NULL;
      }
      else {
            temp=front;
            front=front->next;
            free(temp);
      }
}

void display()
{
```

```
        NODE *temp=front;
        if(rear==NULL)
        {
              printf("queue empty\n");
              return;
        }
        printf("The elements in the queue are\n");
        while(temp!=NULL)
        {
              printf("%d, ", temp->num);
              temp=temp->next;
        }
}
```

Input/Output:

```
Enter the following commands
 '1' to insert
 '2' to delete
 '3' to display
Enter the Command
1
enter the data separated by spaces
23 45 67 78 89
enter 1 to continue
1
Enter the Command
3
The elements in the queue are
23, 45, 67, 78, 89, enter 1 to continue
1
Enter the Command
2
deleted
enter 1 to continue
1
Enter the Command
2
deleted
enter 1 to continue
1
Enter the Command
3
The elements in the queue are
67, 78, 89, enter 1 to continue
0
```