LINKED LISTS

```
/*
 * ******************
 * LINKED LISTS IN C *
 * ******************
 */

#include <stdio.h>
#include <conio.h>
#include <malloc.h>

struct node {
  int data;
  struct node * next;
  struct node * prev;
 } * start, * last;

struct node_ {
  int coef;
  int exp;
  struct node_ * link;
 } * p1_start, * p2_start, * p3_start;

int getdata() {
  int data;
  printf("ENTER DATA : ");
  scanf("%d", & data);
  return data;
}

char menu(char * head) {
  char ch;
  printf("%s\n\n", head);
  printf("MAIN MENU\n\n");
  printf("1. CREATE LIST\n");
  printf("2. INSERT A NODE\n");
  printf("3. DELETE A NODE\n");
  printf("4. DISPLAY\n");
  printf("5. EXIT\n\n");
  printf("PRESS A KEY..\n");
  ch = getch();
  return ch;
}

char insert_menu(struct node * start) {
  if (start == NULL) {
    printf("LIST IS EMPTY!");
    return 0;
  } else {
    printf("\nINSERTION\n");
    printf("1. INSERT AT THE BEGINNING\n");
```

```
    printf("2. INSERT AFTER\n");
    printf("3. INSERT AT THE END\n");
    printf("\nPRESS A KEY..\n");
    return getch();
  }
}

//**********SIMPLE LINKED LISTS**********//

void insert_at_end_simple(int data) {
  struct node * temp, * temp2;
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->next = NULL;
  temp->data = data;
  if (start == NULL) {
    start = temp;
  } else {
    temp2 = start;
    while (temp2->next != NULL) {
      temp2 = temp2->next;
    }
    temp2->next = temp;
  }
}

void insert_at_beg_simple(int data) {
  struct node * temp;
  temp = malloc(sizeof(struct node));
  temp->data = data;
  temp->next = start;
  start = temp;
}

void insert_in_between_simple(int data) {
  int pos;
  struct node * temp, * temp2;
  printf("ENTER POSITION : ");
  scanf("%d", & pos);
  temp2 = start;
  while (--pos) {
    temp2 = temp2->next;
    if (temp2 == NULL) {
      printf("INVALID POSITION!");
      return;
    }
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->data = data;
  temp->next = temp2->next;
  temp2->next = temp;
}

void delete_simple() {
```

```
  int data;
  struct node * temp, * temp2;
  if (start == NULL) {
    printf("LIST IS EMPTY!");
  } else {
    temp = start;
    data = getdata();
    if (data == start->data) {
      start = start->next;
      free(temp);
      return;
    }
    while (temp->next->next != NULL) {
      if (data == temp->next->data) {
     temp2 = temp->next;
     temp->next = temp2->next;
     free(temp2);
     return;
      }
      temp = temp->next;
    }
    if (data == temp->next->data) {
      temp2 = temp->next;
      temp->next = NULL;
      free(temp2);
      return;
    }
    printf("DATA NOT FOUND!");
  }
}
void create_list_simple() {
  int n, i, data;
  printf("ENTER NUMBER OF NODES : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("ENTER DATA FOR NODE %d : ", i + 1);
    scanf("%d", & data);
    insert_at_end_simple(data);
  }
}

void display_simple() {
  struct node * temp;
  if (start == NULL) {
    printf("LIST IS EMPTY!");
    return;
  } else {
    temp = start;
    while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
    }
  }
```

```
}

void simple() {
  int data;
  while (1) {
    clrscr();
    switch (menu("SIMPLE LINKED LISTS")) {
    case '1':
      create_list_simple();
      break;
    case '2':
      switch (insert_menu(start)) {
      case '1':
     insert_at_beg_simple(getdata());
     break;
      case '2':
     insert_in_between_simple(getdata());
     break;
      case '3':
     insert_at_end_simple(getdata());
     break;
      case 0:
     break;
      default:
     printf("WRONG CHOICE!");
      }
      break;
    case '3':
      delete_simple();
      break;
    case '4':
      display_simple();
      break;
    case '5':
      exit(0);
    default:
      printf("WRONG CHOICE!");
    }
    getch();
  }
}

//**********CIRCULAR LINKED LISTS**********//

void display_circu() {
  struct node * temp;
  if (last == NULL) {
    printf("LIST IS EMPTY!");
    return;
  } else {
    temp = last->next;
    while (temp != last) {
      printf("%d ", temp->data);
```

```
      temp = temp->next;
    }
    printf("%d ", temp->data);
  }
}

void delete_circu() {
  int data;
  struct node * temp, * temp2;
  if (last == NULL) {
    printf("LIST IS EMPTY!");
  } else {
    temp = last->next;
    data = getdata();
    if (last == last->next && data == last->data) {
      temp = last;
      last = NULL;
      free(temp);
      return;
    }
    if (data == temp->data) {
      temp2 = temp;
      temp = temp->next;
      last->next = temp;
      free(temp2);
      return;
    }
    while (temp->next != last) {
      if (data == temp->next->data) {
     temp2 = temp->next;
     temp->next = temp2->next;
     free(temp2);
     return;
      }
      temp = temp->next;
    }

    if (data == last->data) {
      temp2 = last;
      temp->next = last->next;
      last = temp;
      free(temp2);
      return;
    }
    printf("DATA NOT FOUND!");
  }
}

void insert_in_between_circu(int data) {
  int pos;
  struct node * temp, * temp2;
  printf("ENTER POSITION : ");
  scanf("%d", & pos);
```

```
  temp2 = last->next;
  while (--pos) {
    temp2 = temp2->next;
    if (temp2 == NULL || temp2 == last) {
      printf("INVALID POSITION!");
      return;
    }
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->data = data;
  temp->next = temp2->next;
  temp2->next = temp;
}

void insert_at_beg_circu(int data) {
  struct node * temp;
  temp = malloc(sizeof(struct node));
  temp->data = data;
  temp->next = last->next;
  last->next = temp;
}

void insert_at_end_circu(int data) {
  struct node * temp;
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->data = data;
  if (last == NULL) {
    last = temp;
    last->next = last;
  } else {
    temp->next = last->next;
    last->next = temp;
    last = temp;
  }
}

void create_list_circu() {
  int n, i, data;
  printf("ENTER NUMBER OF NODES : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("ENTER DATA FOR NODE %d : ", i + 1);
    scanf("%d", & data);
    insert_at_end_circu(data);
  }
}

void circular() {
  int data;
  while (1) {
    clrscr();
    switch (menu("CIRCULAR LINKED LISTS")) {
    case '1':
```

```
      create_list_circu();
      break;
    case '2':
      switch (insert_menu(last)) {
      case '1':
        insert_at_beg_circu(getdata());
        break;
      case '2':
        insert_in_between_circu(getdata());
        break;
      case '3':
        insert_at_end_circu(getdata());
     break;
      case 0:
        break;
      default:
        printf("WRONG CHOICE!");
      }
      break;
    case '3':
      delete_circu();
      break;
    case '4':
      display_circu();
      break;
    case '5':
      exit(0);
    default:
      printf("WRONG CHOICE!");
    }
    getch();
  }
}

//**********DOUBLY LINKED LISTS***********//

void insert_at_end_doubly(int data) {
  struct node * temp, * temp2;
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->next = NULL;
  temp->data = data;
  if (start == NULL) {
    start = temp;
    temp->prev = NULL;
  } else {
    temp2 = start;
    while (temp2->next != NULL) {
      temp2 = temp2->next;
    }
    temp2->next = temp;
    temp->prev = temp2;
  }
}
```

```
void insert_at_beg_doubly(int data) {
  struct node * temp;
  temp = malloc(sizeof(struct node));
  temp->data = data;
  temp->next = start;
  start->prev = temp;
  start = temp;
}

void insert_in_between_doubly(int data) {
  int pos;
  struct node * temp, * temp2;
  printf("ENTER POSITION : ");
  scanf("%d", & pos);
  temp2 = start;
  while (--pos) {
    temp2 = temp2->next;
    if (temp2 == NULL) {
      printf("INVALID POSITION!");
      return;
    }
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->data = data;
  temp->next = temp2->next;
  temp->prev = temp2;
  temp2->next = temp;
  temp->next->prev = temp;
}

void delete_doubly() {
  int data;
  struct node * temp, * temp2;
  if (start == NULL) {
    printf("LIST IS EMPTY!");
  } else {
    temp = start;
    data = getdata();
    if (data == start->data) {
      start = start->next;
      start->prev = NULL;
      free(temp);
      return;
    }
    while (temp->next->next != NULL) {
      if (data == temp->data) {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        free(temp);
        return;
      }
      temp = temp->next;
```

```
    }
    temp = temp->next;
    if (data == temp->data) {
      temp->prev->next = NULL;
      free(temp);
      return;
    }
    printf("DATA NOT FOUND!");
  }
}
void create_list_doubly() {
  int n, i, data;
  printf("ENTER NUMBER OF NODES : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("ENTER DATA FOR NODE %d : ", i + 1);
    scanf("%d", & data);
    insert_at_end_doubly(data);
  }
}

void display_doubly() {
  struct node * temp;
  if (start == NULL) {
    printf("LIST IS EMPTY!");
    return;
  } else {
    temp = start;
    while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
    }
  }
}

void doubly() {
  int data;
  while (1) {
    clrscr();
    switch (menu("DOUBLY LINKED LISTS")) {
    case '1':
      create_list_doubly();
      break;
    case '2':
      switch (insert_menu(start)) {
      case '1':
        insert_at_beg_doubly(getdata());
        break;
      case '2':
        insert_in_between_doubly(getdata());
        break;
      case '3':
        insert_at_end_doubly(getdata());
```

```
        break;
      case 0:
        break;
      default:
        printf("WRONG CHOICE!");
      }
      break;
    case '3':
      delete_doubly();
      break;
    case '4':
      display_doubly();
      break;
    case '5':
      exit(0);
    default:
      printf("WRONG CHOICE!");
    }
    getch();
  }
}

//**********CIRCULAR DOUBLY LINKED LISTS***********//

void display_cdoubly() {
  struct node * temp;
  if (last == NULL) {
    printf("LIST IS EMPTY!");
  } else {
    temp = last->next;
    while (temp != last) {
      printf("%d ", temp->data);
      temp = temp->next;
    }
    printf("%d ", temp->data);
  }
}

void delete_cdoubly() {
  int data;
  struct node * temp, * temp2;
  if (last == NULL) {
    printf("LIST IS EMPTY!");
  } else {
    temp = last->next;
    data = getdata();
    if (last->next == last->prev && data == last->data) {
      temp = last;
      last = NULL;
      free(temp);
      return;
    }
    if (data == temp->data) {
```

```
      temp2 = temp;
      temp = temp->next;
      temp->prev = last;
      last->next = temp;
      free(temp2);
      return;
    }
    while (temp != last) {
      if (data == temp->data) {
     temp2 = temp;
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        free(temp2);
        return;
      }
      temp = temp->next;
    }

    if (data == last->data) {
      temp2 = last;
      last->prev->next = last->next;
      last->next->prev = last->prev;
      last = temp->prev;
      free(temp);
      return;
    }
    printf("DATA NOT FOUND!");
  }
}

void insert_in_between_cdoubly(int data) {
  int pos;
  struct node * temp, * temp2;
  printf("ENTER POSITION : ");
  scanf("%d", & pos);
  temp2 = last->next;
  while (--pos) {
    temp2 = temp2->next;
    if (temp2 == NULL || temp2 == last) {
      printf("INVALID POSITION!");
      return;
    }
  }
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->data = data;
  temp->next = temp2->next;
  temp->prev = temp2;
  temp2->next = temp;
  temp->next->prev = temp;
}

void insert_at_beg_cdoubly(int data) {
  struct node * temp;
```

```
  temp = malloc(sizeof(struct node));
  temp->data = data;
  temp->next = last->next;
  last->next->prev = temp;
  last->next = temp;
  temp->prev = last;
}

void insert_at_end_cdoubly(int data) {
  struct node * temp;
  temp = (struct node * ) malloc(sizeof(struct node));
  temp->data = data;
  if (last == NULL) {
    last = temp;
    last->next = last;
    last->prev = last;
  } else {
    temp->next = last->next;
    temp->prev = last;
    last->next->prev = temp;
    last->next = temp;
    last = temp;
  }
}

void create_list_cdoubly() {
  int n, i, data;
  printf("ENTER NUMBER OF NODES : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    printf("ENTER DATA FOR NODE %d : ", i + 1);
    scanf("%d", & data);
    insert_at_end_cdoubly(data);
  }
}

void cir_doubly() {
  int data;
  while (1) {
    clrscr();
    switch (menu("CIRCULAR DOUBLY LINKED LISTS")) {
    case '1':
      create_list_cdoubly();
      break;
    case '2':
      switch (insert_menu(last)) {
      case '1':
        insert_at_beg_cdoubly(getdata());
        break;
      case '2':
        insert_in_between_cdoubly(getdata());
        break;
      case '3':
```

```
        insert_at_end_cdoubly(getdata());
        break;
      case 0:
     break;
      default:
        printf("WRONG CHOICE!");
      }
      break;
    case '3':
      delete_cdoubly();
      break;
    case '4':
      display_cdoubly();
      break;
    case '5':
      exit(0);
    default:
      printf("WRONG CHOICE!");
    }
    getch();
  }
}

//**********POLYNOMIAL ARITHMETIC**********//

void display(struct node_ * start) {
  struct node_ * temp;
  temp = start;
  while (temp != NULL) {
    printf("(%d)x^%d + ", temp->coef, temp->exp);
    temp = temp->link;
  }
}
struct node_ * input(struct node_ * start) {
  int i, n;
  struct node_ * temp2, * temp;
  printf("HOW MANY TERMS YOU WANT TO INPUT? : ");
  scanf("%d", & n);
  for (i = 0; i < n; i++) {
    temp = malloc(sizeof(struct node_));
    temp->link = NULL;
    printf("ENTER COEFFICIENT FOR TERM %d : ", i);
    scanf("%d", & temp->coef);
    printf("ENTER EXPONENT FOR TERM %d : ", i);
    scanf("%d", & temp->exp);
    if (start == NULL || temp->exp > start->exp) {
      temp->link = start;
      start = temp;
    } else {
      temp2 = start;
      while (temp2->link->exp > temp->exp && temp2 != NULL)
     temp2 = temp2->link;
      temp->link = temp2->link;
```

```
      temp2->link = temp;
    }
    getch();
  }
  return start;
}

struct node_ * add(struct node_ * p1, struct node_ * p2) {
  struct node_ * temp, * p3;
  p3 = p3_start;
  while (p1 != NULL && p2 != NULL) {
    temp = malloc(sizeof(struct node_));
    temp->link = NULL;
    if (p1->exp > p2->exp) {
      temp->exp = p1->exp;
      temp->coef = p1->coef;
      p1 = p1->link;
    } else if (p1->exp < p2->exp) {
      temp->exp = p2->exp;
      temp->coef = p2->coef;
      p2 = p2->link;
    } else if (p1->exp == p2->exp) {
      temp->exp = p1->exp;
      temp->coef = p1->coef + p2->coef;
      p1 = p1->link;
      p2 = p2->link;
    }
    if (p3_start == NULL) {
      p3_start = temp;
      p3 = temp;
    } else {
      p3->link = temp;
      p3 = p3->link;
    }
  }
  while (p1 != NULL) {
    temp = malloc(sizeof(struct node_));
    temp->link = NULL;
    temp->coef = p1->coef;
    temp->exp = p1->exp;
    if (p3 == NULL) {
      p3_start = temp;
      p3 = temp;
    } else p3->link = temp;
    p1 = p1->link;
  }
  while (p2 != NULL) {
    temp = malloc(sizeof(struct node_));
    temp->link = NULL;
    temp->coef = p2->coef;
    temp->exp = p2->exp;
    if (p3 == NULL) {
      p3_start = temp;
```

```
      p3 = temp;
    } else p3->link = temp;
    p2 = p2->link;
  }
  return p3_start;
}

void poly_arth() {
  clrscr();
  printf("POLYNOMIAL ARITHMETIC USING LINKED LISTS\n\n");
  printf("ENTER POLYNOMIAL 1 : \n");
  p1_start = input(p1_start);
  printf("ENTER POLYNOMIAL 2 : \n");
  p2_start = input(p2_start);
  p3_start = add(p1_start, p2_start);
  clrscr();
  display(p1_start);
  display(p2_start);
  printf("=\n");
  display(p3_start);
  getch();
  exit(0);
}

//**********MAIN**********//

void main() {
  char ch;
  while (1) {
    clrscr();
    printf("LINKED LISTS\n\n");
    printf("1. SIMPLE LINKED LISTS\n");
    printf("2. CIRCULAR LINKED LISTS\n");
    printf("3. DOUBLY LINKED LISTS\n");
    printf("4. CIRCULAR DOUBLY LINKED LISTS\n");
    printf("5. APPLICATIONS OF LINKED LISTS\n");
    printf("6. EXIT\n\n");
    printf("PRESS A KEY..\n");
    ch = getch();
    switch (ch) {
    case '1':
      simple();
      break;
    case '2':
      circular();
      break;
    case '3':
      doubly();
      break;
    case '4':
      cir_doubly();
      break;
    case '5':
```

```
      poly_arth();
    case '6':
      exit(0);
    default:
      printf("WRONG CHOICE!");
    }
    getch();
  }
}
```

OUTPUT

MAIN MENU

```
LINKED LISTS

1. SIMPLE LINKED LISTS
2. CIRCULAR LINKED LISTS
3. DOUBLY LINKED LISTS
4. CIRCULAR DOUBLY LINKED LISTS
5. APPLICATIONS OF LINKED LISTS
6. EXIT

PRESS A KEY..
```

SIMPLE LINKED LISTS

```
SIMPLE LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
ENTER NUMBER OF NODES : 5
ENTER DATA FOR NODE 1 : 1
ENTER DATA FOR NODE 2 : 2
ENTER DATA FOR NODE 3 : 3
ENTER DATA FOR NODE 4 : 4
ENTER DATA FOR NODE 5 : 5
```

```
SIMPLE LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..

INSERTION
1. INSERT AT THE BEGINNING
2. INSERT AFTER
3. INSERT AT THE END

PRESS A KEY..
ENTER DATA : 66
```

```
SIMPLE LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
66 1 2 3 4 5 _
```

CIRCULAR LINKED LISTS

```
CIRCULAR LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
ENTER NUMBER OF NODES : 4
ENTER DATA FOR NODE 1 : 1
ENTER DATA FOR NODE 2 : 4
ENTER DATA FOR NODE 3 : 6
ENTER DATA FOR NODE 4 : 8
```

```
CIRCULAR LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
ENTER DATA : 5
DATA NOT FOUND!_
```

```
CIRCULAR LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
1 4 6 8
```

DOUBLY LINKED LISTS

```
DOUBLY LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
ENTER NUMBER OF NODES : 4
ENTER DATA FOR NODE 1 : 33
ENTER DATA FOR NODE 2 : 56
ENTER DATA FOR NODE 3 : 78
ENTER DATA FOR NODE 4 : 34
```

```
DOUBLY LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
33 56 78 34
```

CIRCULAR DOUBLY LINKED LISTS

```
CIRCULAR DOUBLY LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
ENTER NUMBER OF NODES : 5
ENTER DATA FOR NODE 1 : 3
ENTER DATA FOR NODE 2 : 4
ENTER DATA FOR NODE 3 : 5
ENTER DATA FOR NODE 4 : 6
ENTER DATA FOR NODE 5 : 7
```

```
CIRCULAR DOUBLY LINKED LISTS

MAIN MENU

1. CREATE LIST
2. INSERT A NODE
3. DELETE A NODE
4. DISPLAY
5. EXIT

PRESS A KEY..
3 4 5 6 7
```

POLYNOMIAL ARITHMETIC USING LINKED LISTS

```
POLYNOMIAL ARITHMATIC USING LINKED LISTS

ENTER POLYNOMIAL 1 :
HOW MANY TERMS YOU WANT TO INPUT? : 3
ENTER COEFFICIENT FOR TERM 0 : 3
ENTER EXPONENT FOR TERM 0 : 2
ENTER COEFFICIENT FOR TERM 1 : 4
ENTER EXPONENT FOR TERM 1 : 1
ENTER COEFFICIENT FOR TERM 2 : -2
ENTER EXPONENT FOR TERM 2 : 0
ENTER POLYNOMIAL 2 :
HOW MANY TERMS YOU WANT TO INPUT? : 2
ENTER COEFFICIENT FOR TERM 0 : 5
ENTER EXPONENT FOR TERM 0 : 3
ENTER COEFFICIENT FOR TERM 1 : 6
ENTER EXPONENT FOR TERM 1 : 1
```

```
(3)x^2 + (4)x^1 + (-2)x^0 + (5)x^3 + (6)x^1 + =
(5)x^3 + (3)x^2 + (10)x^1 + (-2)x^0 +
```