MỤC LỤC

I. Danh sách liên kết

1. Tạo DSLK:

```
struct nut
{
    int giaTri;
    nut *tiep;
};
typedef nut Node;
Node *dau;
void khoiTao(Node *&dau)
{
    dau=NULL;
}
void nhapDS(Node *&dau)
{
    Node *p, *q;
    int x;
    do{
            cout<<"Nhap mot so nguyen, nhap 0 de dung: ";
            cin>>x;
            if(x!=0)
            {
                    p = new Node;
                    p->giaTri=x;
                    p->tiep=NULL;
                    if(dau==NULL)
                            dau=p;
                    else
                    q->tiep=p;
                    q=p;
            }
    }while(x!=0);
}
```

2. Duyệt danh sách:

```
void duyetDS(Node *dau)
{
    Node *p;
    p=dau;
    while(p!=NULL)
    {
            cout<<fixed<<" "<<p->giaTri;
            p=p->tiep;
    }
}

void duyetDeQuy(Node *dau)
{
    if(dau!=NULL)
    {
            cout<<fixed<<" "<<dau->giaTri;
            duyetDeQuy(dau->tiep);
    }
}
```

3. Các phép toán trên danh sách:

```
int tinhTong(Node *dau)
{
    if(dau==NULL)
            return 0;
    else
            return dau->giaTri + tinhTong(dau->tiep);
}
int tinhTongSoLe(Node *dau)
{
    if(dau==NULL)
            return 0;
    else
            if((dau->giaTri % 2) == 0)
                    return tinhTongSoLe(dau->tiep);
            else
                    return dau->giaTri + tinhTongSoLe(dau->tiep);
}

int demSoChan(Node *dau)
{
    if (dau==NULL)
            return 0;
    else
            if((dau->giaTri % 2) == 0)
                    return 1 + demSoChan(dau->tiep);
            else
                    return demSoChan(dau->tiep);
}
```

4. Sắp xếp danh sách:

```
void sapXep(Node *&dau)
{
    Node *p, *q;
    int tam;
    p=dau;
    while(p->tiep != NULL)
    {
            q=p->tiep;
            while(q!=NULL)
            {
```

```
                if(p->giaTri > q->giaTri)
                {
                        tam=p->giaTri;
                        p->giaTri = q->giaTri;
                        q->giaTri = tam;
                }
                q=q->tiep;
        }
        p=p->tiep;
    }
}
```

5. Thêm phần tử vào danh sách

```
void chenDau(Node *&dau, int x)
{
    Node *p;
    p = new Node;
    p->giaTri = x;
    p->tiep = dau;
    dau = p;
}
void chenThuTu(Node *&dau, int x)
{
    if((dau = NULL)||(dau->giaTri >= x))
            chenDau(dau, x);
    else
            chenThuTu(dau->tiep, x);
}
void chenCuoi(Node *&dau, int x)
{
    Node *p, *q;
    p = new Node;
    p->giaTri = x;
    p->tiep = NULL;
    if(dau == NULL)
            chenDau(dau, x);
    else
    {
            q = dau;
            while(q->tiep != NULL)
                    q = q->tiep;
            q->tiep = p;
    }
}
void chenSau(Node *&dau, int x, int y)
{
    Node *p, *q;
    p = dau;
    while(p != NULL && p->giaTri != y)
    {
            p = p->tiep;
            if(p != NULL)
            {
                    q = new Node;
                    q->giaTri = x;
                    q->tiep = p->tiep;
                    p->tiep =q;
            }
    }
}
void chenTruoc(Node *&dau, int x, int y)
{
    Node *p, *q, *t;
    p = dau;
    while(p != NULL && p->giaTri != y)
    {
            t = p;
            p = p->tiep;
    }
    if(p != NULL)
            if(p == dau)
                    chenDau(dau, x);
            else
            {
                    q = new Node;
                    q->giaTri = x;
                    q->tiep = p;
                    t->tiep = q;
            }
}
```

6. Xóa phần tử khỏi DS

```
void xoaDau(Node *&dau, int x)
{
    Node *p;
    if(dau != NULL)
    {
            p = dau;
            dau = dau->tiep;
            delete p;
    }
}
void xoaCuoi(Node *&dau, int x)
{
    Node *p, *q;
    q = NULL;
    p = dau;
    while((p != NULL) && (p->tiep != NULL))
```

```
    {
            q = p;
            p = p->tiep;
    }
    if(p != NULL)
            if(p == dau)
                    xoaDau(dau, x);
            else
            {
                    q->tiep = NULL;
                    delete p;
            }
}
```

7. Đảo ngược danh sách;

```
void daoNguocTaoDSMoi(Node *dau)
{
    Node *p;
    while(p != NULL)
    {
            chenDau(dau, p->giaTri);
            p = p->tiep;
    }
}
void daoNguoc(Node *&dau) //Khong Tao DS Moi
{
    Node *p, *q;
    q = NULL; p = NULL;
    while(dau != NULL)
    {
            p = dau;
            dau = dau->tiep;
            p->tiep = q;
            q = p;
    }
    dau = p;
}
```

8. Tìm nút giữa danh sách mà chỉ duyệt 1 lần

```
void nutGiua(Node *dau)
{
    Node *p, *q;
    int dem=0;
    p=dau; q=dau->tiep->tiep;
    while(q!=NULL)
        {
            p=p->tiep;
            q=q->tiep->tiep;
            dem++;
        }
        cout<<"\nPhan tu giua danh sach la: "<<p->giaTri;
    }
```

9. Tính tổng các nút có thứ tự lẻ

```
int tongTTLe(Node *dau)
{
        Node *p;
        int dem = 0, sum =0;
        p = dau; dem =1;
        while(p != NULL)
        {
                if(dem % 2 != 0)
                sum += p->giatri;
                p = p->tiep;
                dem++;
        }
        return sum;
}
```

10. Xóa nút trước nút cuối cùng:

```
    void xoaNutTruocNutCuoi(Node *&dau)
    {
        Node *p, *q, *t;
        p = dau;
        q = NULL, t = NULL;
        while(p->tiep != NULL)
        {
                t = q;
                q = p;
                p = p->tiep;
        }
        if(q != NULL)
        {
                if(q == dau)
                {
                        dau = dau->tiep;
                        delete q;
                }
                else
                {
                        t->tiep = p;
                        delete q;
                }
```

```
      }
   }
```

11. Nối hai DSLK

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

struct node {
   int data;
   struct node *next;
};

struct node *even = NULL;
struct node *odd = NULL;
struct node *list = NULL;

//tao danh sach lien ket
void insert(int data) {
   // cap phat bo nho cho node moi;
   struct node *link = (struct node*) malloc(sizeof(struct node));
   struct node *current;

   link->data = data;
   link->next = NULL;

   if(data%2 == 0) {
      if(even == NULL) {
         even = link;
         return;
      }else {
         current = even;

         while(current->next != NULL)
         current = current->next;

         // chen link vao phan cuoi cua list
         current->next = link;
      }
   }else {
      if(odd == NULL) {
         odd = link;
         return;
      }else {
         current = odd;

         while(current->next!=NULL)
            current = current->next;

         // chen link vao phan cuoi cua list
         current->next = link;
      }
   }
}

void display(struct node *head) {
   struct node *ptr = head;

   printf("[head] =>");

   while(ptr != NULL) {
      printf(" %d =>",ptr->data);
      ptr = ptr->next;
   }

   printf(" [null]\n");
}

void combine() {
   struct node *link;

   list = even;
   link = list;

   while(link->next!= NULL) {
      link = link->next;
   }

   link->next = odd;
}

int main() {
   int i;

   for(i=1; i<=10; i++)
      insert(i);

   printf("Danh sach chan: ");
   display(even);

   printf("Danh sach le: ");
   display(odd);
```

```
   combine();

   printf("Sau khi noi: \n");
   display(list);

   return 0;
}
```

12. Chia đôi DSLK

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

struct node {
   int data;
   struct node *next;
};

struct node *even = NULL;
struct node *odd = NULL;
struct node *list = NULL;

//tao danh sach lien ket
void insert(int data) {
   // cap phat bo nho cho node moi;
   struct node *link = (struct node*) malloc(sizeof(struct node));
   struct node *current;

   link->data = data;
   link->next = NULL;

   if(list == NULL) {
      list = link;
      return;
   }
   current = list;

   while(current->next!=NULL)
      current = current->next;

   // chen link vao phan cuoi cua list
   current->next = link;
}

void display(struct node *head) {
   struct node *ptr = head;

   printf("[head] =>");
   //bat dau tu phan dau cua list
   while(ptr != NULL) {
      printf(" %d =>",ptr->data);
      ptr = ptr->next;
   }

   printf(" [null]\n");
}

void split_list() {
   // cap phat bo nho cho node moi;
   struct node *link;
   struct node *current;

   while(list != NULL) {
      struct node *link = (struct node*) malloc(sizeof(struct node));

      link->data = list->data;
      link->next = NULL;

      if(list->data%2 == 0) {
         if(even == NULL) {
            even = link;
            list = list->next;
            continue;
         }else {
            current = even;

            while(current->next != NULL)
            current = current->next;

            // chen link vao phan cuoi cua list
            current->next = link;
         }
         list = list->next;
      }else {
         if(odd == NULL) {
            odd = link;
            list = list->next;
            continue;
         }else {
            current = odd;
```

```
            while(current->next!=NULL)
            current = current->next;

            // chen link vao phan cuoi cua list
            current->next = link;

         }
         list = list->next;
      }
   }
}

int main() {
   int i;

   for(i=1; i<=10; i++)
      insert(i);

   printf("Danh sach ban dau: \n");
   display(list);

   split_list();

   printf("\nDanh sach le: ");
   display(odd);

   printf("\nDanh sach chan: ");
   display(even);

   return 0;
}
```

13. Tạo DSLK vòng

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;

//chen link tai vi tri dau tien
void insert(int data) {
   // cap phat bo nho cho node moi;
   struct node *link = (struct node*) malloc(sizeof(struct node));

   link->data = data;
   link->next = NULL;

   // neu head la trong, tao list moi
   if(head==NULL) {
      head = link;
      head->next = link;
      return;
   }

   current = head;

   // di chuyen toi phan cuoi list
   while(current->next != head)
      current = current->next;

   // chen link vao phan cuoi cua list
   current->next = link;

   // lien ket last node voi head
   link->next = head;

}

//hien thi list
void printList() {
   struct node *ptr = head;

   printf("\n[head] =>");

   //bat dau tu phan dau cua list
   while(ptr->next != head) {
      printf(" %d =>",ptr->data);
      ptr = ptr->next;
   }

   printf(" %d =>",ptr->data);
   printf(" [head]\n");
}
```

```
int main() {
   insert(10);
   insert(20);
   insert(30);
   insert(1);
   insert(40);
   insert(56);

   printList();
   return 0;
}
```

14. Tạo DSLK đôi

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

struct node {
   int data;
   struct node *prev;
   struct node *next;
};

struct node *head = NULL;
struct node *last = NULL;

struct node *current = NULL;

//tao danh sach lien ket
void insert(int data) {
   // cap phat bo nho cho node moi;
   struct node *link = (struct node*) malloc(sizeof(struct node));

   link->data = data;
   link->prev = NULL;
   link->next = NULL;

   // neu head la trong, tao list moi
   if(head==NULL) {
      head = link;
      return;
   }

   current = head;

   // di chuyen toi phan cuoi list
   while(current->next!=NULL)
      current = current->next;

   // chen link vao phan cuoi cua list
   current->next = link;
   last = link;
   link->prev = current;
}

//hien thi list
void printList() {
   struct node *ptr = head;

   printf("\n[head] <=>");

   //bat dau tu phan dau cua list
   while(ptr->next != NULL) {
      printf(" %d <=>",ptr->data);
      ptr = ptr->next;
   }

   printf(" %d <=>",ptr->data);
   printf(" [head]\n");
}

int main() {
   insert(10);
   insert(20);
   insert(30);
   insert(1);
   insert(40);
   insert(56);

   printList();
   return 0;
}
```

15. Quản lý văn bản:

```
#include <string.h>
struct nut
{
       char Data[80];
       nut *sau,*truoc;
};
typedef nut Node;
```

```
Node *dau;
Node *dau1;
int i;
void khoitao(Node *&dau)
{
	dau=NULL;
}
void nhapvb(Node *&dau)
{
	char tam[80];
	Node *q,*p;
	do
	{
		printf("Nhap vao dong van ban, Enter dung: ");
		fflush(stdin);
		gets(tam);
		if(strcmp(tam,"\0")!=0)
		{
			p=new Node;
			strcpy(p->Data,tam);
			p->sau=NULL;
			p->truoc=NULL;
			if(dau==NULL)
			dau=p;
			else
			{
				q->sau=p;
				p->truoc=q;
			}
			q=p;
		}

	}
	while(strcmp(tam,"\0")!=0);
}
void duyetvb(Node *dau)
{
	if(dau!=NULL)
	{
		printf("\n%s",dau->Data);
		duyetvb(dau->sau);
	}
}
void indongi(Node *dau, int i,int dem=1)
{
	if(dau!=NULL)
	{
		if(dem==i)
		printf("\n%s",dau->Data);
		else
		indongi(dau->sau,i,dem+1);
	}
}
void indongi2(Node *dau,int i)
{
	int dem;
	Node *p;
	p=dau;
	dem=1;
	while((p!=NULL)&&(dem!=i))
	{
		p=p->sau;
		dem=dem+1;
	}
	if(p!=NULL)
	printf("\ndong %d la:\n%s",i,p->Data);
}
void inij(Node *dau, int i, int j)
{
	int dem;
	Node *p,*q;
	if(j>=i)
	{
// tim i
		dem=1;
		p=dau;
		while(p!=NULL&&dem!=i)
		{
			p=p->sau;
			dem=dem+1;
		}
		if(p!=NULL)
		{
// tim j
			q=p;
			while(q!=NULL&&dem!=j)
			{
				q=q->sau;
				dem=dem+1;
			}
```

```
// in ra dong i den dong j.
                        if(q!=NULL)
                        while(p!=q->sau)
                        {
                                printf("\n%s",p->Data);
                                p=p->sau;
                        }
                }

        }
}
void xoadongi(Node *&dau,int i)
{
        Node *p,*q,*t;
        int dem;
        p=dau; dem=1;
        while(p!=NULL&&dem!=i)
        {
                p=p->sau;
                dem=dem+1;
        }
        if(p!=NULL)
        {
                if(p==dau)
                {
                        dau=dau->sau;
                        dau->truoc=NULL;
                        delete p;
                }
                else
                if(p->sau==NULL)
                {
                        q=p->truoc;
                        q->sau=NULL;
                        delete p;
                }
                else
                {
                        q=p->truoc;
                        t=p->sau;
                        q->sau=t;
                        t->truoc=q;
                        delete p;
                }
        }
}
void xoadongidq(Node *&dau,int i,int dem=1)
{
        Node *p;
        if(dau!=NULL)
        {
                if(dem==i)
                {
                        p=dau;
                        dau=dau->sau;
                        dau->truoc=NULL;
                        delete p;
                }
                else
                xoadongidq(dau->sau,i,dem+1);
        }
}
void chepij(Node *dau, int i, int j, Node *&dau1)
{
        int dem;
        Node *p,*q,*t,*l;
        if(j>=i)
        {
// tim i
                dem=1;
                p=dau;
                while(p!=NULL&&dem!=i)
                {
                        p=p->sau;
                        dem=dem+1;
                }
                if(p!=NULL)
                {
// tim j
                        q=p;
                        while(q!=NULL&&dem!=j)
                        {
                                q=q->sau;
                                dem=dem+1;
                        }
// in ra dong i den dong j.
                        if(q!=NULL)
                        while(p!=q->sau)
                        {
                                //printf("\n%s",p->Data);
```

```
                        t=new Node;
                        strcpy(t->Data,p->Data);
                        t->sau=NULL;t->truoc=NULL;
                            if (dau1==NULL)
                              dau1=t;
                              else
                              {
                               l->sau=t;
                               t->truoc=l;
                                 }
                                 l=t;
                        p=p->sau;
                    }
                }

        }
}
void chendong(Node *&dau,int i, int j, int k)
{
        int m;
        Node *v,*u,*w;
        v=dau;
        m=1;
        chepij(dau,i,j,dau1);
        while(v!=NULL&&m!=k)
        {
                v=v->sau;
                m=m+1;
        }
        if(v!=NULL)
        {   // v tro den dong k
                u=dau1;
                while(u->sau!=NULL)
                u=u->sau;
                // u la dong cuoi cua ds dau1

                w=v->sau; // dong sau dong k
                v->sau=dau1;
                dau1->truoc=v;
                u->sau=w;
                w->truoc=u;
        }
}
void xoaij(Node *&dau,int i, int j)
{
        Node *p,*q,*u,*v;
        int dem;
        if(i<j)
        {
                dem=1;
                p=dau;
                while(p!=NULL&&dem!=i)
                {
                        p=p->sau;

                        dem=dem+1;
                }
                q=p;
                if(p!=NULL)
                {
                        while(q!=NULL&&dem!=j)
                        {
                                q=q->sau;
                                dem=dem+1;
                        }
                        if(q!=NULL)
                        {
                                u=p->truoc;
                                u->sau=q->sau;
                                q->sau=v; v->truoc=u;
                                while(p!=q)
                                {
                                        delete p;
                                        p=p->sau;
                                }

                        }
                }
        }
}
```

16. Quản lý hộ khẩu:

```
        #include <iostream>
        #include <string.h>
        using namespace std;
        //1. khai báo dữ liệu
        struct xe
        {
                char soXe[9];
                char hieuXe[30];
                xe *tiep;
```

```
};
struct con
{
	char MSCN[7];
	char hoTenC[30];
	con *noi;
};

struct HK //Hộ khẩu
{
	char soHK[9];
	char hoTenChuHo[30];
	con *conl;
	xe *xel;
	HK *sau;
};

//2. Nhập dữ liệu
// Máy gõ soHK gõ enter kết thúc... ngược lại nhập dữ liệu.
// máy yc gõ MSCN, gõ enter kết thúc, ngược lại nhập dữ liệu
// máy yc nhập soXe, enter kết thúc, ngược lại nhập dữ liệu

HK *dauhk;
void khoiTao(HK *&dauhk)
{
	dauhk = NULL;
}

void nhapXe(xe *&dau)
{
	xe *p, *q;
	char soXeTmp[9];
	do
	{
		cout<<"Nhap so xe, go enter de dung: ";
		fflush(stdin);
		gets(soXeTmp);
		if(strcmp(soXeTmp, "\0") != 0)
		{
			p = new xe;
			strcpy(p->soXe, soXeTmp);
			cout<<"Nhap hieu xe: ";
			fflush(stdin);
			gets(p->hieuXe);
			p->tiep == NULL;
			if(dau == NULL)
				dau = p;
			else
				q->tiep = p;
			q = p;
		}
	}while(strcmp(soXeTmp, "\0") != 0);
}

void nhapCon(con *&dau)
{
	con *p, *q;
	char MSCNtmp[7];
	do
	{
		/* code */
	}while();
}

void qlkh(HK *&dauhk)
{
	HK *p, *q;
	char soHKtmp[9];
	do
	{
		cout<<"Nhap so ho khau: ";
		fflush(stdin);
		gets(soHKtmp);
		if(strcmp(soHKtmp, "\0") != 0)
		{
			p = new HK;
			strcpy(p->soHK, soHKtmp);
			cout<<"Nhap ho ten chu ho: ";
			fflush(stdin);
			gets(p->hoTenChuHo);
			p->xel = NULL;
			p->conl = NULL;
			p->sau = NULL;
			nhapCon(p->conl);
			nhapXe(p->xel);
			if(dauhk = NULL)
				dau = p;
			else
				q->sau = p;
			q = p;
```

```
                }
        }while(strcmp(soHKtmp, "\0") != 0);
}
/*
1. hiển thị soHk, Tên chủ hộ, đếm số con
*/

int demcon(con *x)
{
        if(x == NULL)
                return 0;
        else
                return 1 + demcon(x->noi);
}
void hienthi(HK *dauhk)
{
        if(dauhk != NULL)
        {
                cout<<dauhk->soHK<<" "<<dauhk->hoTenChuHo;
                cout<<"So con: "<<demcon(dauhk->conl);
                hienthi(dauhk->sau);
        }
}
int main()
{
        /* code */
        return 0;
}
```

17. Quản lý ôtô

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct xe
{
        char soxe[8];
        char loaixe[15];
        xe *tiep;};
struct canhan
{char MSCN[5];
 char Hoten[30];
 xe *xel;
 canhan *noi;};
canhan *daucn;
void khoitao(canhan *&daucn)
{daucn=NULL;}
void nhapxe(xe *&dau)
{
    xe *p,*q;
    char soxet[8];
    do{
       printf("nhap so xe: ");
       fflush(stdin);
       gets(soxet);
       if(strcmp(soxet,"\0")!=0)
       {
              p=new xe;
              strcpy(p->soxe,soxet);
              printf("loai xe: ");
              fflush(stdin);
              gets(p->loaixe);
              p->tiep=NULL;
              if(dau==NULL)
                 dau=p;
              else
                 q->tiep=p;
                 q=p;
              }
       }
       while(strcmp(soxet,"\0")!=0);
    }
void nhapdl(canhan *&daucn)
{
        canhan *p,*q;
        char MSCNt[5];
        do{
                printf("MSCN: ");
                fflush(stdin);
                gets(MSCNt);
                if(strcmp(MSCNt,"\0")!=0)
                {
                        p=new canhan;
                        strcmp(p->MSCN,MSCNt);
                        printf("Ho ten: ");
                        fflush(stdin);
```

```
                        gets(p->Hoten);
                        p->xel=NULL;
                        p->noi=NULL;
                        nhapxe(p->xel);
                        if(daucn==NULL)
                           daucn=p;
                        else
                        q->noi=p;
                        q=p;
                }
        }
        while(strcmp(MSCNt,"\0")!=0);
}
//In ra MSCN, Hoten tung nguoi
void duyetxe(xe *dau)
{
        if(dau!=NULL)
           {
                printf("\n %s %s",dau->soxe,dau->loaixe);
                duyetxe(dau->tiep);
           }

}
void duyetcanhan(canhan *daucn)
{
        if(daucn!=NULL)
        {
                printf("\n %s %s",daucn->MSCN,daucn->Hoten);
                duyetxe(daucn->xel);
                duyetcanhan(daucn->noi);
        }
}
//in ra ds nhung nguoi co sd xe loai honda
int honda(xe *dau)
{
        if(dau==NULL)
           return 0;
        else
            if(strcmp(dau->loaixe,"Honda")==0)
               return 1;
            else
               return honda(dau->tiep);
}
void duyethonda(canhan *daucn)
{
        if(daucn!=NULL)
        {
                if(honda(daucn->xel)==1)
                   printf("\n %s %s",daucn->MSCN,daucn->Hoten);
                duyethonda(daucn->noi);
        }
}
//in ra nguoi co 2 chiec honda
int main()
{
        khoitao(daucn);
        nhapdl(daucn);
        duyetcanhan(daucn);
        printf("\nnguoi co sd xe honda la:");
        duyethonda(daucn);
        return 0;
}
```

II. Cây nhị phân:

1. Tạo cây

```
struct nut
{
    int info;
    nut *left, *right;
};
typedef nut Node;
Node *root, *root1;
void khoiTao(Node *&root)
{
    root = NULL;
}
void mocNut(Node *&root, int x)
{
    if(root == NULL)
    {
            root = new Node;
            root->info = x;
            root->left = NULL;
            root->right = NULL;
    }
    else
    {
            if(root->info >= x)
                    mocNut(root->left,x);
            else
                    mocNut(root->right,x);
    }
}

void taoCay(Node *&root)
{
    int tmp;
    do
    {
            cout<<"Nhap so nguyen, 0 de dung: ";
            cin>>tmp;
            if(tmp != 0)
                    mocNut(root,tmp);
    }while(tmp != 0);
}
/*Trong main()
khoiTao(root);
taoCay(root);
*/
```

2. Duyệt cây LNR

```
void duyetLNR(Node *root)
{
    if(root != NULL)
    {
            duyetLNR(root->left);
            cout<<root->info<<" ";
            duyetLNR(root->right);
    }
}
```

3. Duyệt cây RLN

```
void duyetRLN(Node *root)
{
    if(root != NULL)
    {
            duyetRLN(root->right);
            duyetRLN(root->left);
            cout<<root->info<<" ";
    }
}
```

4. Duyệt cây NRL

```
void duyetNRL(Node *root)
{
    if(root != NULL)
    {
                    cout<<root->info<<" ";
            duyetNRL(root->right);
            duyetNRL(root->left);
    }
}
```

5. Đếm số nút, tính tổng các nút:

```
int demSoNut(Node *root)
{
    if(root ==NULL)
            return 0;
    return 1 + demSoNut(root->left) + demSoNut(root->right);
}
int tongNut(Node *root)
{
```

```
    if(root == NULL)
            return 0;
    if(root->left == NULL && root->right == NULL)
            return root->info;
    return root->info + tongNut(root->left) + tongNut(root->right);
}
```

6. Đếm số lá, in các giá trị lá, tính tổng lá

```
int demLa(Node *root)
{
    if(root == NULL)
            return 0;
    if((root->left == NULL) && (root->right == NULL))
            return 1;
    return demLa(root->left) + demLa(root->right);
}
void inLa(Node *root)
{
    if(root != NULL)
            if((root->left == NULL) && (root->right == NULL))
                    cout<<root->info<<" ";
            else
            {
                    inLa(root->left);
                    inLa(root->right);
            }
}
int tongLa(Node *root)
{
    if(root == NULL)
            return 0;
    if((root->left == NULL) && (root->right == NULL))
            return root->info;
    return tongLa(root->left) + tongLa(root->right);
}
```

7. Đếm các nút chẵn:

```
int demChan(Node *root)
{
    if(root == NULL)
            return 0;
    if((root->info % 2) == 0)
            return 1 + demChan(root->left) + demChan(root->right);
    return  demChan(root->left) + demChan(root->right);
}
```

8. Chiều cao của cây

```
int max(int a, int b)
{
    if(a>=b)
            return a;
    return b;
}
int chieuCao(Node *root)
{
    if(root == NULL)
            return 0;
    return 1 + max(chieuCao(root->left), chieuCao(root->right));
}
```

9. Mức thứ i, tính tổng giá trị ở mức thứ i:

```
void inMuci(Node *root, int i, int muc = 1)
{
    if(root != NULL)
            if(muc == i)
                    cout<<root->info<<" ";
            else
            {
                    inMuci(root->left, i, muc+1);
                    inMuci(root->right, i, muc+1);
            }
}
int tongMuci(Node *root, int i, int muc = 1)
{
    if(root == NULL)
            return 0;
    if(muc == i)
            return root->info;
    return tongMuci(root->left, i, muc+1) + tongMuci(root->right, i,
muc+1);
}
```

10. Tìm vị trí của phần tử mang giá trị x

```
Node *tim(Node *root, int x)
{
    if(root == NULL)
            return NULL;
    if(root->info = x);
            return root;
    if(root->info > x)
            return tim(root->left, x);
```

```
    return tim(root->right, x);
}
```

11. Đếm số nút có tổng cây con trái lớn hơn tổng cây con phải

```
int demNutTongTraiPhai(Node *root)
{
    if(root == NULL)
            return 0;
    if ((tongNut(root->left)) > (tongNut(root->right)))
            return 1 +     demNutTongTraiPhai(root->left) +
demNutTongTraiPhai(root->right);
    return demNutTongTraiPhai(root->left) + demNutTongTraiPhai(root-
>right);
}
```

12. Sao chép cây nhị phân

```
void saoChep(Node *root, Node *&root1)
{
    if(root == NULL)
            root1 = NULL;
    else
    {
            root1 = new Node;
            root1->info = root->info;
            saoChep(root->left, root1->left);
            saoChep(root->right, root1->right);
    }
}
/*
main()
khoiTao(root1);
saoChep(root, root1);
*/
```

13. Đếm gốc cây con trái mang giá trị chẵn, gốc cây con phải mang giá trị lẻ

```
int demLcRl(Node *root)
{
    if(root == NULL)
            return 0;
    if(root->left == NULL || root->right == NULL)
            return demLcRl(root->left) + demLcRl(root->right);
    if(root->left->info %2 == 0 && root->right->info %2 !=0)
            return 1 + demLcRl(root->left) + demLcRl(root->right);
    return demLcRl(root->left) + demLcRl(root->right);
}
```

14. Đếm nút có chiều cao cây con trái và chiều cao cây con phải sai kém nhau quá 1 đơn vị:

```
    int max(int a, int b);
    int chieuCao(Node *root);
    int demChieuCaoLR(Node *root)
    {
        if(root == NULL)
                return 0;
        if(abs(chieuCao(root->left) - chieuCao(root->right)) > 1 )
                return 1 + demChieuCaoLR(root->left) + demChieuCaoLR(root-
    >right);
        return demChieuCaoLR(root->left) + demChieuCaoLR(root->right);
    }
```

15. Kiểm tra cây có phải cây nhị phân tìm kiếm không.

```
int KTcayNPTK(Node *root)
{
    if (root == NULL)
            return 1;
    if(root->left == NULL && root->right == NULL)
            return 1;
    else
            if(root->left == NULL) KTcayNPTK(root->right);
            else
                    if(root->right == NULL) KTcayNPTK(root->left);
                    else
                            if(root->left->info > root->info || root-
>right->info < root->info)
                                    return 0;
}
```

16. Cây nhị phân sinh viên – Student Binary Tree

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

    struct item {
        char id[20];
        char name[20];
        double scores;
    };
    struct Node {
        item key; //truong key cua du lieu
        Node *Left, *Right; //con trai va con phai
    };
    typedef Node *Tree;  //cay
```

```
int compare(item x, item y) { // so sanh 2 item theo key
    return strcmp(x.id, y.id);
}

item inputItem() {  // nhap 1 item
    item x;
    char id[20];
    printf("Enter id of student (q to quit): ");
    gets(x.id);

    if (strcmp(x.id, "q") == 0 || strcmp(x.id, "Q") == 0) {
        return x;
    }

    printf("Enter name of student: ");
    gets(x.name);

    printf("Enter scores of student:");
    scanf("%lf", &x.scores);

    //fflush(stdin);
    while (getchar() != '\n');

    return x;
}

void outItem(item x) {
    printf("%-20s %-20s %-3.2f \n", x.id, x.name, x.scores);
}

int insertNode(Tree &T, item x) // chen 1 Node vao cay
        {
    if (T != NULL) {
        if (compare(T->key, x) == 0)
            return -1;  // Node nay da co
        if (compare(T->key, x) > 0)
            return insertNode(T->Left, x); // chen vao Node trai
        else if (compare(T->key, x) < 0)
            return insertNode(T->Right, x);   // chen vao Node phai
    }
    T = (Node *) malloc(sizeof(Node));
    if (T == NULL)
        return 0;    // khong du bo nho
    T->key = x;
    T->Left = T->Right = NULL;
    return 1;   // ok
}
```

```
void CreateTree(Tree &T)        // nhap cay
        {
    item x;
    while (1) {
        printf("Enter a student: ");
        x = inputItem();
        if (strcmp(x.id, "q") == 0 || strcmp(x.id, "Q") == 0)
            break;  // x = 0 thi thoat
        int check = insertNode(T, x);
        if (check == -1)
            printf("Student is exits!");
        else if (check == 0)
            printf("Memory full");

    }
}

// Duyet theo LNR
void LNR(Tree T) {
    if (T != NULL) {
        LNR(T->Left);
        outItem(T->key);
        LNR(T->Right);
    }
}

Node* searchScores(Tree T, int scores)     // tim nut co diem < 4
        {
    if (T != NULL) {
        if (T->key.scores < scores) {
            Node *P = T;
            return P;
        }
        if (T->key.scores >= scores) {
            Node *node = searchScores(T->Left, scores);
            if (node != NULL)
                return node;
            else {
                return searchScores(T->Right, scores);
            }
        }
    }
    return NULL;
}

int delKey(Tree &T, item x)     // xoa nut co key x
        {
    if (T == NULL)
        return 0;
```

```
    else if (compare(T->key, x) > 0)
        return delKey(T->Left, x);
    else if (compare(T->key, x) < 0)
        return delKey(T->Right, x);
    else // T->key == x
    {
        Node *P = T;
        if (T->Left == NULL)
            T = T->Right;    // Node chi co cay con phai
        else if (T->Right == NULL)
            T = T->Left;   // Node chi co cay con trai
        else // Node co ca 2 con
        {
            Node *S = T, *Q = S->Left;
            // S la cha cua Q, Q la Node phai nhat cua cay con trai
cua P
            while (Q->Right != NULL) {
                S = Q;
                Q = Q->Right;
            }
            P->key = Q->key;
            S->Right = Q->Left;
            delete Q;
        }
    }
    return 1;
}

int main() {
    Tree T;
    T = NULL; //Tao cay rong

    CreateTree(T); //Nhap cay
    //duyet cay
    printf("LNR: \n");
    LNR(T);
    printf("\n");

    // them sinh vien
    item x;
    printf("Enter id of student to add: ");
    x = inputItem();
    if (insertNode(T, x) == -1) {
        printf("add failt!");
    } else {
        printf("add success:\n");
        printf("LNR after add item: \n");
        LNR(T);
        printf("\n");
    }

    // xoa sinh vien co diem nho hon 4
    int del = 1;
    while (del) {
        Node* node = searchScores(T, 4);
        if (node != NULL) {
            printf("del");
            del = delKey(T, node->key);
        } else {
            printf("null");
            del = 0;
        }
    }
    printf("LNR after delete item: \n");
    LNR(T);
    printf("\n");
    return 0;
}
```

III. 18 bài toán về danh sách liên kết

1. Count()

Viết hàm Count() thực hiện việc đếm các lần xuất hiện của một số nguyên trong một danh sách.

```
void CountTest() {
	List myList = BuildOneTwoThree(); // build {1, 2, 3}
	int count = Count(myList, 2); // returns 1 since there's 1 '2' in the list
}
```

Mẫu hàm:

```
/* Given a list and an int, return the number of times that int occurs in the list. */
int Count(struct node* head, int searchFor) {
	// Your code
}
```

2. Nth()

Viết hàm Nth() trả về đối tượng thứ n trong danh sách (bắt đầu từ 0)

```
void GetNthTest() {
	struct node* myList = BuildOneTwoThree(); // build {1, 2, 3}
	int lastNode = GetNth(myList, 2); // returns the value 3
}
```

Mẫu hàm:

```
// Given a list and an index, return the data
// in the nth node of the list. The nodes are numbered from 0.
// Assert fails if the index is invalid (outside 0..lengh-1).
int GetNth(struct node* head, int index) {
	// Your code
}
```

3. DeleteList()

Viết hàm DeleteList() để xóa một danh sách liên kết và thiết lập con trỏ head thành NULL

```
void DeleteListTest() {
	struct node* myList = BuildOneTwoThree(); // build {1, 2, 3}
	DeleteList(&myList); // deletes the three nodes and sets myList to NULL
}
```

Mẫu hàm:

```
void DeleteList(struct node** head) {
	// Your code
}
```

4. Pop()

Viết hàm Pop() để lấy ra phần tử đầu tiên của danh sách, phần từ này đồng thời được xóa khỏi danh sách.

```
void PopTest() {
	struct node* head = BuildOneTwoThree(); // build {1, 2, 3}
	int a = Pop(&head); // deletes "1" node and returns 1
	int b = Pop(&head); // deletes "2" node and returns 2
	int c = Pop(&head); // deletes "3" node and returns 3
	int len = Length(head); // the list is now empty, so len == 0
}
```

Mẫu hàm:

```
/*
The opposite of Push(). Takes a non-empty list
and removes the front node, and returns the data
which was in that node.
*/
int Pop(struct node** headRef) {
	// your code...
}
```

5. InsertNth()

Viết hàm để thêm vào danh sách một đối tượng tại vị trí thứ n

```
void InsertNthTest() {
	struct node* head = NULL; // start with the empty list
	InsertNth(&head, 0, 13); // build {13)
	InsertNth(&head, 1, 42); // build {13, 42}
	InsertNth(&head, 1, 5); // build {13, 5, 42}
	DeleteList(&head); // clean up after ourselves
}
```

Mẫu hàm:

```
/*
A more general version of Push().
Given a list, an index 'n' in the range 0..length,
and a data element, add a new node to the list so
that it has the given index.
*/
void InsertNth(struct node** headRef, int index, int data) {
	// your code...
}
```

6. SortedInsert()

Có một danh sách đã được sắp xếp. Viết hàm để thêm vào danh sách một đối tượng vào danh sách và giữ được thứ tự sắp xếp.
Mẫu hàm:
```
void SortedInsertNth(struct node** headRef,struct node *newNode) {
	// your code
}
```

7. InsertSort()

Nhận một danh sách đầu vào, sắp xếp danh sách này theo thứ tự tăng dần, bài toán yêu cầu sử dụng hàm SortedInsert() ở câu 6.
Mẫu hàm:
```
// Given a list, change it to be in sorted order (using SortedInsert()).
void InsertSort(struct node** headRef) { // Your code
}
```

8. Append()

Viết hàm nhận hai tham số là danh sách A và B, trả về danh sách A với B được gắn vào đuôi của A và B được thiết lập là NULL.
Mẫu hàm:
```
// Append 'b' onto the end of 'a', and then set 'b' to NULL.
void Append(struct node** aRef, struct node** bRef) {
	// Your code...
}
```

9. FronBackSplit()

Viết hàm tách một danh sách liên kết ra làm hai, ngắt ở giữa. Nếu chiều dài của danh sách là lẻ thì danh sách thứ nhất sẽ dài hơn danh sách thứ hai một phần tử.
Mẫu hàm:
```
/*
Split the nodes of the given list into front and back halves,
and return the two lists using the reference parameters.
If the length is odd, the extra node should go in the front list.
*/
void FrontBackSplit(struct node* source,
struct node** frontRef, struct node** backRef) {
	// Your code...
}
```

10. RemoveDuplicates()

Cho một danh sách đã được sắp xếp theo thứ tự tăng dần, hãy xóa các phần tử bị lặp với điều kiện danh sách chỉ được duyệt một lần.
Mẫu hàm:
```
/*
Remove duplicates from a sorted list.
*/
void RemoveDuplicates(struct node* head) {
	// Your code...
}
```

11. MoveNode()

Mẫu hàm:
```
/*
Take the node from the front of the source, and move it to
the front of the dest.
It is an error to call this with the source list empty.
*/
void MoveNode(struct node** destRef, struct node** sourceRef) {
	// Your code
}
```

12. ShuffleMerge()

Mẫu hàm:
```
/*
Merge the nodes of the two lists into a single list taking a node
alternately from each list, and return the new list.
*/
struct node* ShuffleMerge(struct node* a, struct node* b) {
	// Your code
}
```

Lời giải

1. Count()
```
int Count(struct node* head, int searchFor) {
	int count =0;
	struct node *element = head;
	while (element) {
		if ( element->data == searchFor) count++;
		element = element-> next;
	}
	return count;
}
```

2. Nth()

```
int GetNth(struct node* head, int index) {
	struct node *elem = head;
	int i = 0;

	while (elem) {
		if (i==index)
			return elem->data;
		elem = elem->next;
		i++;
	}
	assert(0);
}
```

3. DeleteList()

```
void DeleteList(struct node **head) {
	struct node *elem = *head;
	struct node *another;
	while (elem) {
		another = elem->next;
		free(elem);
		elem = another;
	}
		*head = NULL;
}
```

4. Pop()

```
int Pop(struct node **head) {
	struct node *elem = *head;
	int a;
	assert( elem );
	a = elem->data;
	*head = elem->next;
	free(elem);
	return a;
}
```

5. InsertNth()

```
void InsertNth(struct node** head, int index, int data) {
	struct node *newElem;
	newElem = (struct node*) malloc(sizeof(struct node));
	newElem->data = data;
	int i = 0;
	struct node *elem = *head;
	while (elem) {
		if (i== (index-1)) {
			newElem->next = elem->next;
			elem->next = newElem;
			break;
		}
		i++;
		elem = elem->next;
		}
}
```

6. SortedInsert()

```
void InsertNth(struct node** head, truct node *newNode) {
	struct node *elem = *head;
	while (elem) {
		if (newNode->data < elem->data) {
			newNode->next = elem->next;
			elem->next = newNode;
			break;
		}
		elem = elem->next;
		}
}
```

7. InsertSort()

```
void InsertNth(struct node** head, truct node *newNode) {
}
```

8. Append()

```
void Append(struct node** aRef, struct node** bRef) {
		struct node *elem = *aRef;
		if (*aRef == NULL) {
			*aRef = *bRef;
		} else {
			while (elem->next) {
				elem = elem->next;
			}
			elem->next = *bRef;
		}
		*bRef = NULL;
}
```

9. FronBackSplit()

```
void FrontBackSplit(struct node* source,
		struct node** frontRef, struct node** backRef) {
		if (source == NULL || source->next == NULL) {
			*frontRef = source;
			*backRef = NULL;
		} else {
			struct node *fast = source;
			struct node *slow = source->next;
			while (fast) {
				fast = fast->next;
				if (fast != NULL) {
					fast = fast->next;
```

```
                slow = slow->next;
            }
        }
        *frontRef = source;
        *backRef = slow->next;
        slow->next = NULL;
    }
}
```