```
    #include <stdio.h>
    #include <stdlib.h>
    #include <conio.h>
    #define MAX_TREE_HT 100
    struct MH_Node
    {
      char character;
      unsigned frequency;
      struct MH_Node *l, *r;
    };


    struct M_Heap
    {
      unsigned size;
      unsigned space;
      struct MH_Node **array;
    };

    struct MH_Node* newNode(char character, unsigned frequency)
    {
        struct MH_Node* temp = (struct MH_Node*) malloc(sizeof(struct MH_Node));
        temp->l = temp->r = NULL;
        temp->character = character;
        temp->frequency = frequency;
        return temp;
    }


    struct M_Heap* createM_Heap(unsigned space)
    {
        struct M_Heap* M_Heap = (struct M_Heap*) malloc(sizeof(struct M_Heap));
        M_Heap->size = 0;
        M_Heap->space = space;
        M_Heap->array = (struct MH_Node**)malloc(M_Heap->space * sizeof(struct
MH_Node*));
        return M_Heap;
    }


    void swapMH_Node(struct MH_Node** a, struct MH_Node** b)
    {
        struct MH_Node* t = *a;
        *a = *b;
        *b = t;
    }


    void M_Heapify(struct M_Heap* M_Heap, int idx)
    {
        int smallest = idx;
        int l = 2 * idx + 1;
        int r = 2 * idx + 2;

        if (l < M_Heap->size && M_Heap->array[l]->frequency < M_Heap-
>array[smallest]->frequency)
            smallest = l;

      if (r < M_Heap->size && M_Heap->array[r]->frequency < M_Heap-
>array[smallest]->frequency)
            smallest = r;

        if (smallest != idx)
        {
```

```
            swapMH_Node(&M_Heap->array[smallest], &M_Heap->array[idx]);
            M_Heapify(M_Heap, smallest);
        }
    }

    int isSizeOne(struct M_Heap* M_Heap)
    {
        return (M_Heap->size == 1);
    }


    struct MH_Node* extractMin(struct M_Heap* M_Heap)
    {
        struct MH_Node* temp = M_Heap->array[0];
        M_Heap->array[0] = M_Heap->array[M_Heap->size - 1];
        --M_Heap->size;
        M_Heapify(M_Heap, 0);
        return temp;
    }


    void insertM_Heap(struct M_Heap* M_Heap, struct MH_Node* MH_Node)
    {

        int i = M_Heap->size - 1;
        ++M_Heap->size;
        while (i && MH_Node->frequency < M_Heap->array[(i - 1)/2]->frequency)
        {
            M_Heap->array[i] = M_Heap->array[(i - 1)/2];
            i = (i - 1)/2;
        }
        M_Heap->array[i] = MH_Node;
    }


    void buildM_Heap(struct M_Heap* M_Heap)
    {
        int n = M_Heap->size - 1;
        int i;
        for (i = (n - 1) / 2; i >= 0; --i)
            M_Heapify(M_Heap, i);
    }


    void printArr(int arr[], int n)
    {
        int i;
        for (i = 0; i < n; ++i)
            printf("%d", arr[i]);
            printf("\n");
    }


    int isLeaf(struct MH_Node* root)
    {
        return !(root->l) && !(root->r) ;
    }


    struct M_Heap* createAndBuildM_Heap(char character[], int frequency[], int
size)
    {
      int i;
        struct M_Heap* M_Heap = createM_Heap(size);
```

```
        for (i = 0; i < size; ++i)
            M_Heap->array[i] = newNode(character[i], frequency[i]);
        M_Heap->size = size;
        buildM_Heap(M_Heap);
        return M_Heap;
    }


    struct MH_Node* buildHuffmanTree(char character[], int frequency[], int
size)
    {
        struct MH_Node *l, *r, *top;
        struct M_Heap* M_Heap = createAndBuildM_Heap(character, frequency,
size);

        while (!isSizeOne(M_Heap))
        {
            l = extractMin(M_Heap);
            r = extractMin(M_Heap);
            top = newNode('$', l->frequency + r->frequency);
            top->l = l;
            top->r = r;
            insertM_Heap(M_Heap, top);
        }
        return extractMin(M_Heap);
    }

    void printCodes(struct MH_Node* root, int arr[], int top)
    {
        if (root->l)
        {
            arr[top] = 0;
            printCodes(root->l, arr, top + 1);
        }

        if (root->r)
        {
            arr[top] = 1;
            printCodes(root->r, arr, top + 1);
        }

        if (isLeaf(root))
        {
            printf("%c: ", root->character);
            printArr(arr, top);
        }
    }

    void HuffmanCodes(char character[], int frequency[], int size)
    {
      struct MH_Node* root = buildHuffmanTree(character, frequency, size);
      int arr[MAX_TREE_HT], top = 0;
      printCodes(root, arr, top);
    }

    void main()
    {
      char arr[] = {'a', 'b', 'c', 'd', 'e', 'f'};
      int frequency[] = {5, 9, 12, 13, 16, 45};
      int size;
      clrscr();
      size = sizeof(arr)/sizeof(arr[0]);
      HuffmanCodes(arr, frequency, size);
      getch();
```

```
    }
```

Danh sach lien ket don

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

typedef int ElementType;
struct _PointerType{
  ElementType Inf;
  struct _PointerType *Next;
};
typedef struct _PointerType PointerType;

PointerType *InsertMiddle(PointerType *Prev, ElementType X)
{
  PointerType *TempNode;

  TempNode = (PointerType *)malloc(sizeof(PointerType));
  TempNode->Inf = X;
  TempNode->Next = Prev->Next;
  Prev->Next = TempNode;

  return TempNode;
}

ElementType Delete(PointerType *Prev){
  ElementType X;
  PointerType *TempNode;

  TempNode = Prev->Next; Prev->Next = Prev->Next->Next;
  X = TempNode->Inf;
  free(TempNode);

  return X;
}

PointerType *InsertToHead(PointerType *First, ElementType X){
  PointerType *TempNode;

  TempNode = (PointerType *) malloc(sizeof(PointerType));
  TempNode->Inf = X;
  TempNode->Next = First;
  First = TempNode;

  return First;
}

PointerType *InsertToLast(PointerType *First, ElementType X){
  PointerType *NewNode; PointerType *TempNode;

  NewNode = (PointerType *)malloc(sizeof(PointerType));
  NewNode->Inf = X;NewNode->Next = NULL;
  TempNode = First;

  while(TempNode->Next!=NULL)
    TempNode = TempNode->Next;

  TempNode->Next = NewNode;
  return First;
}

PointerType *DeleteHead(PointerType *First){
```

```
  PointerType *TempNode;

  TempNode = First->Next;
  free(First);

  return TempNode;
}

PointerType *DeleteLast(PointerType *First){
  PointerType *Temp1,*Temp2;
  Temp1 = First; Temp2 = First;

  while(Temp1->Next != NULL){
   Temp2 = Temp1;
   Temp1 = Temp1->Next;}

  Temp2->Next = NULL;
  free(Temp1);
  return First;
}

int IsEmpty(PointerType *First)
{
  return !First;
}

PointerType *MakeNull(PointerType *First)
{
  while(!IsEmpty(First))
   First=DeleteHead(First);
  return First;
}

void Print(PointerType *First){
  PointerType *TempNode;

  printf("%p ",First);
  TempNode = First;
  while(TempNode!=NULL){
   printf("%d:%p ",TempNode->Inf,TempNode->Next);
   TempNode = TempNode->Next;
  }
  printf("\n");
}

// Than chuong trinh chinh
int main(){
    PointerType *ds=NULL,*pv=NULL;
    int i,p;
    // Chen 6 gia tri vao vi tri dau
    ds = InsertToHead(ds,1);ds = InsertToHead(ds,2);ds = InsertToHead(ds,3);
    ds = InsertToHead(ds,4);ds = InsertToHead(ds,5);ds = InsertToHead(ds,6);
    // In ra danh sach sau khi chen
    printf("In ra danh sach sau khi chen vao dau \n");
    Print(ds);

    // Chen so 7 vao vi tri cuoi cua danh sach
    printf("Danh sach sau khi chen 7 vao cuoi \n");
    ds = InsertToLast(ds,7);
    Print(ds);

    // In ra danh sach sau khi xoa phan tu dau
    printf("In ra danh sach sau khi xoa dau \n");
    ds = DeleteHead(ds);
```

```
    Print(ds);

    // In ra danh sach sau khi xoa phan tu cuoi
    printf("In ra danh sach sau khi xoa cuoi \n");
    ds = DeleteLast(ds);
    Print(ds);

    // In ra danh sach sau khi xoa phan tu dau
    printf("Cho vi tri ban muon chen 7 vao sau p = ");scanf("%d",&p);
    pv = ds;i=1;
    while(i<p){pv = pv->Next;i++;}
    InsertMiddle(pv,7);
    Print(ds);
    /**/


    getch();
    return 0;
}

// Tính chiều cao cây

int ChieuCao(Tree c)
{
 if (c!=NULL)
 {
   int a = ChieuCao(c->pLeft);
   int b = ChieuCao(c->pRight);
   int max = (a>b)?a:b;
 return 1 + max;
 }
 return 0;
}

//892 Tính tổng các nút có đúng hai con mà thông tin nút đó là số chính phương
bool SoChinhPhuong(int n)
{
    int a = sqrt((double)n);
    if (a*a != n)
        return false;
    return true;
}
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if (SoChinhPhuong(c->iX))
            if (c->pLeft != NULL && c->pRight != NULL)
                return c->iX + a + b;
        return a + b;
    }
    return 0;
}


//891 Tính tổng các nút có đúng một con mà thông tin nút đó là số nguyên tố
bool SoNguyenTo(int n)
{
    if (n<=1)
        return false;
    for (int i=2; i<n; i++)
```

```
        if(n%i==0)
            return false;
    return true;
}
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if (SoNguyenTo(c->iX))
            if ((c->pLeft != NULL && c->pRight == NULL) && (c->pLeft == NULL &&
c->pRight != NULL))
                return c->iX + a + b;
        return a + b;
    }
    return 0;
}

//890 Tính tổng các nút lá mà thông tin tại nút đó là giá trị chẵn
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if (c->iX % 2 == 0)
            if (c->pLeft == NULL && c->pRight == NULL)
                return c->iX + a + b;
        return a + b;
    }
    return 0;
}

//889 Tính tổng các nút lẻ
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if (c->iX % 2 != 0)
            return c->iX + a + b;
        return a + b;
    }
    return 0;
}

int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if (c->pLeft != NULL && c->pRight != NULL)
            return c->iX + a + b;
        return a + b;
    }
    return 0;
}
```

```
//887 Tính tổng các nút có đúng 1 con trong cây
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if ((c->pLeft != NULL && c->pRight == NULL) || (c->pLeft == NULL && c-
>pRight != NULL))
            return c->iX + a + b;
        return a + b;
    }
    return 0;
}

//886 Tính tổng các nút lá trong cây
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        if (c->pLeft == NULL && c->pRight == NULL)
            return c->iX + a + b;
        return a + b;
    }
    return 0;
}

//885 Tính tổng các nút trong cây
int Tinh(Tree c)
{
    if (c!=NULL)
    {
        int a = Tinh(c->pLeft);
        int b = Tinh(c->pRight);
        return c->iX + a + b;
    }
    return 0;
}

//884 Đếm số lượng nút nằm ở tầng cao hơn tầng thứ k trên cây
int ChieuCaoCay(Tree c)
{
    if (c!=NULL)
    {
        int a = ChieuCaoCay(c->pLeft);
        int b = ChieuCaoCay(c->pRight);
        int max = (a>b)?a:b;
        return 1 + max;
    }
    return 0;
}
int DemTangThuk(Tree c, int k)
{
    if (c!=NULL)
    {
        k--;
        int a = DemTangThuk(c->pLeft,k);
        int b = DemTangThuk(c->pRight,k);
        if (k==0)
            return 1 + a + b;
        return a + b;
```

```
    }
    return 0;
}
int Dem(Tree c, int k)
{
    if (c!=NULL)
    {
        int DemSoLuong = 0;
        for (int i=k;i<ChieuCaoCay(c); i++)
        {
            DemSoLuong += DemTangThuk(c,i);
        }
        return DemSoLuong;
    }
    return 0;
}

//883 Đếm số lượng nút nằm ở tầng thấp hơn tầng thứ k trên cây
int DemTangThuk(Tree c, int k)
{
    if (c!=NULL)
    {
        k--;
        int a = DemTangThuk(c->pLeft,k);
        int b = DemTangThuk(c->pRight,k);
        if (k==0)
            return 1 + a + b;
        return a + b;
    }
    return 0;
}
int Dem(Tree c, int k)
{
    if (c!=NULL)
    {
        int DemSoLuong = 0;
        for (int i=1;i<k; i++)
        {
            DemSoLuong += DemTangThuk(c,i);
        }
        return DemSoLuong;
    }
    return 0;
}

//882 Đếm số lượng nút trên tầng thứ k
int Dem(Tree c, int k)
{
    if (c!=NULL)
    {
        k--;
        int a = Dem(c->pLeft,k);
        int b = Dem(c->pRight,k);
        if (k==0)
            return 1 + a + b;
        return a + b;
    }
    return 0;
}
```

```
//881 Đếm số lượng nút có đúng 2 con mà thông tin tại đó là số chính phương
bool SoChinhPhuong(int n)
{
    if (n<=0)
        return 0;
    int s = sqrt((double)n);
    if (s*s == n)
            return 1;
    return 0;
}
int Dem(Tree c)
{
    if (c!=NULL)
    {
        int a = Dem(c->pLeft);
        int b = Dem(c->pRight);
        if (SoChinhPhuong(c->iX))
            if (c->pLeft!=NULL && c->pRight!=NULL)
                return 1 + a + b;
        return a + b;
    }
    return 0;
}

//880 Đếm số lượng nút có đúng 1 con mà thông tin tại đó là số nguyên tố
bool SoNguyenTo(int n)
{
    if (n<=1)
        return 0;
    for (int i=2; i<n; i++)
        if (n%i == 0)
            return 0;
    return 1;
}
int Dem(Tree c)
{
    if (c!=NULL)
    {
        int a = Dem(c->pLeft);
        int b = Dem(c->pRight);
        if (SoNguyenTo(c->iX))
            if ((c->pLeft!=NULL && c->pRight==NULL) || (c->pLeft==NULL && c-
>pRight!=NULL))
                return 1 + a + b;
        return a + b;
    }
    return 0;
}

//879 Đếm số lượng nút lá mà thông tin tại nút đó là giá trị chẵn
int Dem(Tree c)
{
    if (c!=NULL)
    {
        int a = Dem(c->pLeft);
        int b = Dem(c->pRight);
        if (c->iX%2==0 && c->pLeft == NULL && c->pRight == NULL)
            return 1 + a + b;
        return a + b;
    }
    return 0;
}
```

```
//878 Đếm số lượng nút chẵn
int Dem(Tree c)
{
    if (c!=NULL)
    {
        int a = Dem(c->pLeft);
        int b = Dem(c->pRight);
        if (c->iX%2==0)
            return 1 + a + b;
        return a + b;
    }
    return 0;
}


//877 Đếm số lượng nút có đúng 2 con
int Dem(Tree c)
{
    if (c!=NULL)
    {
        int a = Dem(c->pLeft);
        int b = Dem(c->pRight);
        if (c->pLeft != NULL && c->pRight != NULL)
            return 1 + a + b;
        return a + b;
    }
    return 0;
}

//876 Đếm số lượng nút có đúng 1 con
int Dem(Tree c)
{
    if (c!=NULL)
    {
        int a = Dem(c->pLeft);
        int b = Dem(c->pRight);
        if ((c->pLeft != NULL && c->pRight == NULL) || (c->pLeft == NULL && c-
>pRight != NULL))
            return 1 + a + b;
        return a + b;
    }
    return 0;
}

//875* Viết hàm xuất các nút trên cây theo thứ tự tầng 0 đến tầng h-1 của cây
(với h là chiều cao của cây)
int ChieuCaoCay(Tree c)
{
    if (c == NULL)
        return 0;
    int a = ChieuCaoCay(c->pLeft);
    int b = ChieuCaoCay(c->pRight);
    int max = (a>b)?a:b;
    return 1 + max;
}

void XuatTheoTangK(Tree c, int k)
{
    if (c!=NULL)
    {
        k--;
        if (c->pLeft != NULL)
            XuatTheoTangK(c->pLeft,k);
```

```
        if (k==0)
            printf(\"%4d\", c->iX);
        if (c->pRight != NULL)
            XuatTheoTangK(c->pRight,k);
    }
}
void Xuat(Tree c)
{
    int h = ChieuCaoCay(c);
    printf (\"\nChieu cao cay: %d\",h);
    for (int i=0; i<=h-1; i++)
    {
        printf(\"\n tang %d :\", i);
        XuatTheoTangK(c,i+1);
    }
}

//874* Viết hàm xuất các nút trên tầng thứ k của cây
void Xuat(Tree c, int k)
{
    if (c!=NULL)
    {
        k--;
        if (c->pLeft != NULL)
            Xuat(c->pLeft,k);
        if (k==0)
            printf(\"%4d\", c->iX);
        if (c->pRight != NULL)
            Xuat(c->pRight,k);
    }
}
void NhapK(int &k)
{
    printf(\"\nNhap tang thu k: \");
    scanf_s(\"%d\",&k);
}
void main()
{
    Tree c = NULL;
    Nhap(c);
    int k;
    NhapK(k);
    printf(\"\nXuat cac gia tri tang thu k cua cay theo LNR: \");
    Xuat(c,k+1);
}

//871 Viết hàm xuất các giá trị chẵn trong cây
void Xuat(Tree c)
{
    if (c!=NULL)
    {
        if (c->pLeft != NULL)
            Xuat(c->pLeft);
        if (c->iX % 2 == 0)
            printf("%4d", c->iX);
        if (c->pRight != NULL)
            Xuat(c->pRight);
    }
}


//870 Viết hàm xuất các giá trị trong cây
void Xuat(Tree c)
```

```
{
    if (c!=NULL)
    {
        if (c->pLeft != NULL)
            Xuat(c->pLeft);
        printf("%4d", c->iX);
        if (c->pRight != NULL)
            Xuat(c->pRight);
    }
}
```