*#include* <stdio.h>

*#include* "arraydin.h"

*#include* <stdlib.h>

*/\*\**

*\* Konstruktor*

*\* I.S. sembarang*

*\* F.S. Terbentuk ArrayDin kosong dengan ukuran InitialSize*

*\*/*

ArrayDin MakeArrayDin(){

*// ArrayDin arr;*

*// arr.A = (ElType \*) malloc (InitialSize \* sizeof ElType);*

    ArrayDin arr;

    arr.A = (ElType \*) malloc ((InitialSize) \* sizeof (ElType));

    arr.Capacity = InitialSize;

    arr.Neff = 0;

*return* arr;

}

*/\*\**

*\* Destruktor*

*\* I.S. ArrayDin terdefinisi*

*\* F.S. array->A terdealokasi*

*\*/*

void DeallocateArrayDin(ArrayDin \*array){

    free((\*array).A);

}

*/\*\**

*\* Fungsi untuk mengetahui apakah suatu array kosong.*

*\* Prekondisi: array terdefinisi*

*\*/*

boolean IsEmpty(ArrayDin array){

*return* array.Neff == 0;

}

*/\*\**

*\* Fungsi untuk mendapatkan banyaknya elemen efektif array, 0 jika tabel kosong.*

*\* Prekondisi: array terdefinisi*

*\*/*

int Length(ArrayDin array){

*return* array.Neff;

}

*/\*\**

*\* Mengembalikan elemen array L yang ke-I (indeks lojik).*

*\* Prekondisi: array tidak kosong, i di antara 0..Length(array).*

*\*/*

ElType Get(ArrayDin array, IdxType i){

*return* array.A[i];

}

*/\*\**

*\* Fungsi untuk mendapatkan kapasitas yang tersedia.*

*\* Prekondisi: array terdefinisi*

*\*/*

int GetCapacity(ArrayDin array){

*return* array.Capacity;

}

*/\*\**

*\* Fungsi untuk menambahkan elemen baru di index ke-i*

*\* Prekondisi: array terdefinisi, i di antara 0..Length(array).*

*\*/*

void InsertAt(ArrayDin \*array, ElType el, IdxType i){

    int j;

*if* ((\*array).Neff < (\*array).Capacity)

    {

*for* (j = (\*array).Neff; j >= i + 1; j--)

        {

            (\*array).A[j] = (\*array).A[j - 1];

        }

        (\*array).A[i] = el;

        (\*array).Neff += 1;

    }

*else*

    {

        ElType \*temparray = (ElType \*)malloc((\*array).Capacity\* sizeof(ElType));

*for* (j = 0; j < (\*array).Capacity; j++)

        {

            temparray[j] = (\*array).A[j];

        }

        free((\*array).A);

        (\*array).A = (ElType \*)malloc(((\*array).Capacity \* 2 ) \*sizeof(ElType));

*for* (j = 0; j < i; j++)

        {

            (\*array).A[j] = temparray[j];

        }

        (\*array).A[i] = el;

*for* (j = i; j < (\*array).Capacity; j++)

        {

            (\*array).A[j + 1] = temparray[j];

        }

        free(temparray);

        (\*array).Capacity \*= 2;

        (\*array).Neff += 1;

    }

}

*/\*\**

*\* Fungsi untuk menambahkan elemen baru di akhir array.*

*\* Prekondisi: array terdefinisi*

*\*/*

void InsertLast(ArrayDin \*array, ElType el){

    InsertAt((array),el,(\*array).Neff);

}

*/\*\**

*\* Fungsi untuk menambahkan elemen baru di awal array.*

*\* Prekondisi: array terdefinisi*

*\*/*

void InsertFirst(ArrayDin \*array, ElType el){

   InsertAt((array),el,0);

}

*/\*\**

*\* Fungsi untuk menghapus elemen di index ke-i ArrayDin*

*\* Prekondisi: array terdefinisi, i di antara 0..Length(array).*

*\*/*

void DeleteAt(ArrayDin \*array, IdxType i){

    int j;

    ElType e = Get(\*array, i);

*if* ((\*array).Neff > ((\*array).Capacity / 4))

    {

        (\*array).Neff -= 1;

*for* (j = i; j < (\*array).Neff; j++)

        {

            (\*array).A[j] = (\*array).A[j + 1];

        }

    }

*else*

    {

        int Bcapacity = InitialSize;

*if* (Bcapacity < ((\*array).Capacity / 2)) Bcapacity = (\*array).Capacity / 2;

        ElType \*temparray = (ElType \*)malloc(Bcapacity \* sizeof(ElType));

*for* (j = 0; j < i; j++)

        {

            temparray[j] = (\*array).A[j];

        }

        (\*array).Neff -= 1;

*for* (j = i; j < (\*array).Neff; j++)

        {

            temparray[j] = (\*array).A[j + 1];

        }

        free((\*array).A);

        (\*array).A = (ElType \*)malloc(Bcapacity \* sizeof(ElType));

*for* (j = 0; j < (\*array).Neff; j++)

        {

            (\*array).A[j] = temparray[j];

        }

        (\*array).Capacity = Bcapacity;

        free(temparray);

    }

}

*/\*\**

*\* Fungsi untuk menghapus elemen terakhir ArrayDin*

*\* Prekondisi: array tidak kosong*

*\*/*

void DeleteLast(ArrayDin \*array){

    DeleteAt(array,(\*array).Neff-1);

}

*/\*\**

*\* Fungsi untuk menghapus elemen pertama ArrayDin*

*\* Prekondisi: array tidak kosong*

*\*/*

void DeleteFirst(ArrayDin \*array){

    DeleteAt(array,0);

}

*/\*\**

*\* Fungsi untuk melakukan print suatu ArrayDin.*

*\* Print dilakukan dengan format: [elemen-1, elemen-2, ..., elemen-n]*

*\* dan diakhiri newline.*

*\* Prekondisi: array terdefinisi*

*\*/*

void PrintArrayDin(ArrayDin array){

    int j;

    printf("[");

*for* (j=0;j<(array).Neff;j++){

        printf("%d",(array).A[j]);

*if* (j!= (array).Neff-1){

            printf(", ");

        }

    }

    printf("]\n");

}

*/\*\**

*\* Fungsi untuk melakukan reverse suatu ArrayDin.*

*\* Prekondisi: array terdefinisi*

*\*/*

void ReverseArrayDin(ArrayDin \*array){

    int j;

    ElType \*temp = (ElType\*) malloc ((\*array).Capacity \* sizeof(ElType));

*for* (j = 0; j < (\*array).Neff; j++) {

        temp[j] = (\*array).A[(\*array).Neff - (1+j)];

    }

*for* (j = 0; j < (\*array).Neff; j++) {

        (\*array).A[j] = temp[j];

    }

    free(temp);

}

*/\*\**

*\* Fungsi untuk melakukan copy suatu ArrayDin.*

*\* Prekondisi: array terdefinisi*

*\*/*

ArrayDin CopyArrayDin(ArrayDin array){

    ArrayDin coparr;

    IdxType j;

    coparr.A = (ElType\*) malloc (array.Capacity \* sizeof(ElType));

    coparr.Capacity = array.Capacity; coparr.Neff = array.Neff;

*for* (j = 0; j < coparr.Neff; j++) {

        coparr.A[j] = array.A[j];

    }

*return* coparr;

}

*/\*\**

*\* Fungsi untuk melakukan search suatu ArrayDin.*

*\* Index pertama yang ditemukan akan dikembalikan.*

*\* Jika tidak ditemukan, akan mengembalikan -1.*

*\* Prekondisi: array terdefinisi*

*\*/*

IdxType SearchArrayDin(ArrayDin array, ElType el){

    int j =0;

*while* (((array).A[j] != el) && (j <= (array).Neff)) {

        j++;

    }

*if* (j > (array).Neff) {

        j = -1;

    }

*return* j;

}