T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

KONU: MERGE IMPLEMENTATION ON LINKED LISTS

VERİ YAPILARI VE ALGORİTMALAR

1.ÖDEV

BAHAR-(2016-2017)

Ders Adı:
VERİ YAPILARI VE ALGORİTMALAR

Ders Öğretmeni:
Doç. Dr. Mine Elif KARSLIGİL

Hazırlayan:
Adı : MUHAMMED YASİN
Soyadı : SAĞLAM
Numarası: 15011804

TARİH (08.03.2017)

1.YÖNTEM

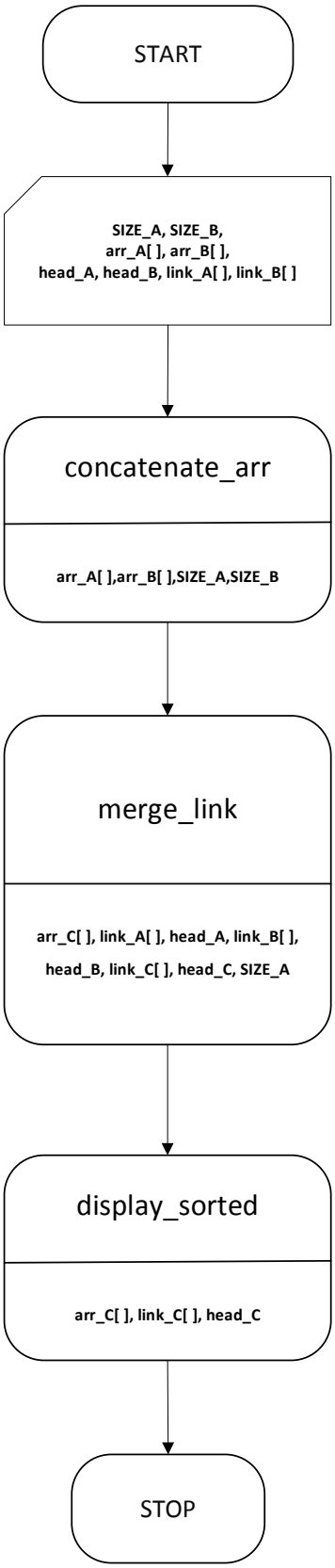
Verilen problemde iki adet dizi, link dizileri ve head değerleri ile birlikte kullanıcıdan okunacaktır. Okuma işlemleri gerçekleştikten sonra 1. Adım olarak kullanıcıdan okunan 2 ayrı dizi tek bir dizi olarak uç uca ekleme yapılarak (concatenate) birleştirilecektir. Birleştirme işlemi tamamlandıktan sonra ilk verilen değer içerikli dizilere erişim olmaksızın, okunan link dizileri ve head değerleri kullanılarak yeni dizinin link dizisinin oluşturulması gerekmektedir. Bu problem algoritma olarak sırasız verilen iki dizinin, uç uca eklenerek yeni bir dizide oluşturulması ve link dizileri yardımıyla merge yapılarak yeni bir link dizisinin oluşturulması olarak da özetlenebilir. Problemin çözümünde 3 adet modül kullanılmştır.

1- Kod kalabalığını ve karmaşasını önlemek için dizi yazdırma fonksiyonu,
2- Dizileri uc uca ekleme işlemini gerçekleştirecek olan concatenate_arr fonksiyonu,
3- Verilen link dizileri üzerinden yeni dizi değerlerine erişim sağlanarak 2 ayrı link dizisindeki değerlerin kıyaslanıp belirlenen koşula göre (küçükten büyüğe olacak şekilde) yeni dizi için link dizisine yapılacak değer atamasını gerçekleyen merge_link fonksiyonu.

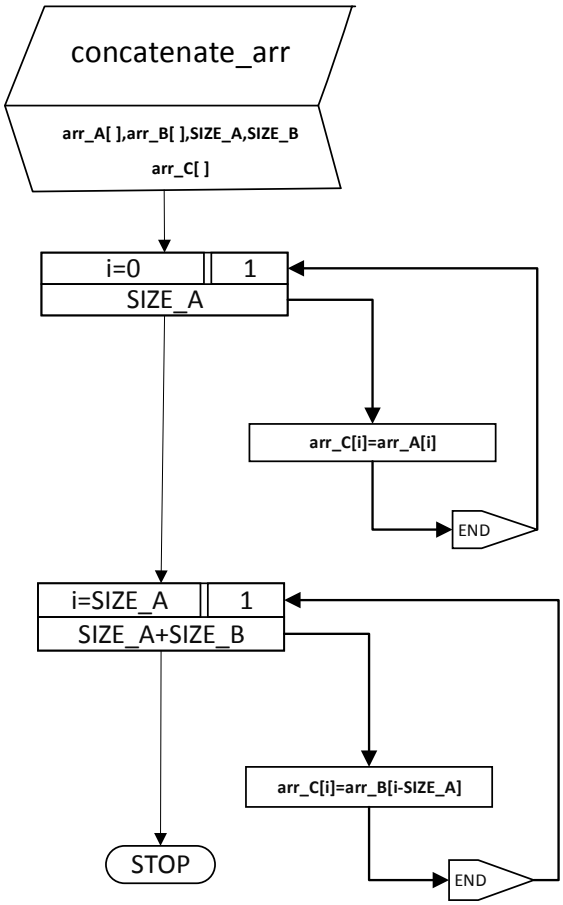
Algoritma olarak en çok karmaşıklığa sahip olan merge_link fonksiyonunda ilk tanımlanan dizi değerlerine yeni dizi üzerinden link dizileri kullanarak erişilmesi gerektiği için yapılacak olan işlemlerde head bilgilerini tutmak için gerekli olan yeni dizinin head değerini belirlemek için bir ön kontrol yapılmaktadır. Ayrıca iterasyon için kullanılacak olan (concatenate işlemi yapıldığı için verilen ilk dizi left, ikinci dizi right olarak düşünüldüğünde) l_next ve r_next değişkenleri default olarak head değerlerini içerecek şekilde atanmışlardır. Yeni dizinin head indexini belirlemek için gerekli kontroller yapıldıktan sonra yeni dizinin head değişkenine değer ataması yapılmış ve yeni link dizisinin geçerli konumunu belirten cur(current) değişkenine de sonraki erişim için gerekli olan değer atanmıştır. Sonrasında ise merge sort algoritmasındaki merge modülüne benzer olacak şekilde while döngüleri kullanılarak (durma noktası l_next veya r_next değişkenlerinin -1 değerini göstermeleri) gerekli kıyaslama işlemleri yapılarak ve iterasyon değişkenleri(l_next, r_next, cur) ilerletilerek yeni link dizisine gereken değer atamaları yapılmıştır. Dizilerin boyutlarının farklı olması ve başlangıçtaki verilen iki diziden birinin tüm değerlerinin atamasının yapılmış olması ihtimaline karşın ilk while döngüsünden çıkıldıktan sonra tekrardan while döngüleri açılarak verilen dizilerde eleman kaldıysa onlarda ilk while döngüsündeki adımlar tekrarlanarak link dizisine atanmaktadır. Son olarak ise yeni link dizisinin en son kalınan konum değerini tutan cur değişkeni kullanılarak link dizisinin en büyük elemanın -1 değerine sahip olan indekse linklenmesi sağlanarak yeni link dizisi tamamlanmaktadır.

AKIŞ DİYAGRAMLARI

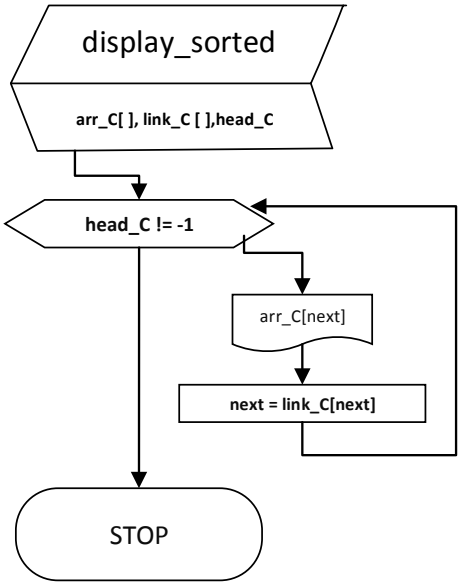
MAIN



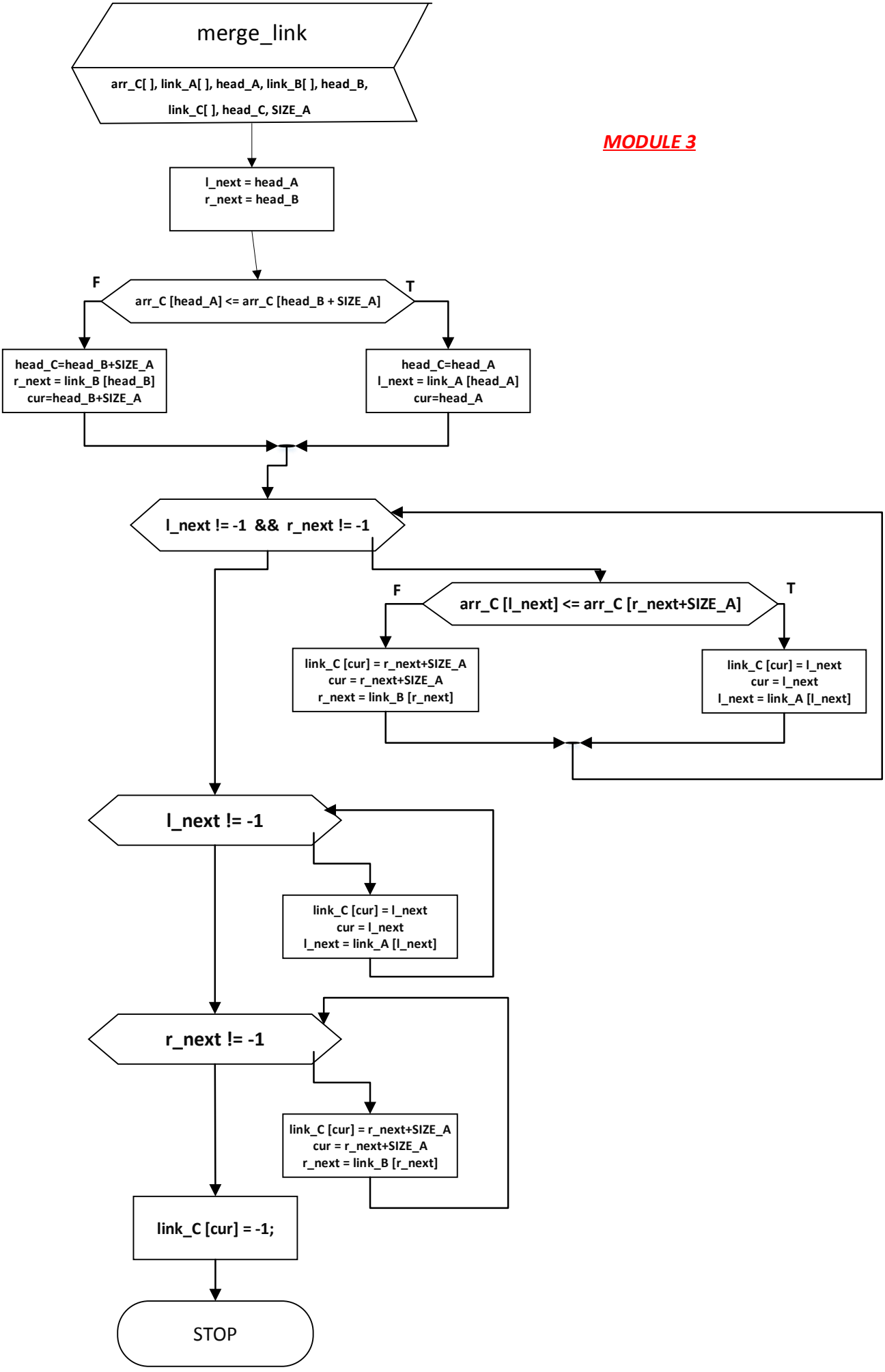
MODULE 1



MODULE 2



MODULE 3


merge_link
arr_C[ ], link_A[ ], head_A, link_B[ ], head_B,
link_C[ ], head_C, SIZE_A
l_next = head_A
r_next = head_B
F
arr_C [head_A] <= arr_C [head_B + SIZE_A]
T
head_C=head_B+SIZE_A
r_next = link_B [head_B]
cur=head_B+SIZE_A
head_C=head_A
l_next = link_A [head_A]
cur=head_A
l_next != -1 && r_next != -1
F
arr_C [l_next] <= arr_C [r_next+SIZE_A]
T
link_C [cur] = r_next+SIZE_A
cur = r_next+SIZE_A
r_next = link_B [r_next]
link_C [cur] = l_next
cur = l_next
l_next = link_A [l_next]
l_next != -1
link_C [cur] = l_next
cur = l_next
l_next = link_A [l_next]
r_next != -1
link_C [cur] = r_next+SIZE_A
cur = r_next+SIZE_A
r_next = link_B [r_next]
link_C [cur] = -1;
STOP

2.UYGULAMA

ANALİZ

ITERATION	CURRENT	L_NEXT	ARR_C [L-NEXT]	LINK_A[L_NEXT]	SIZE_A+R_NEXT	ARR_C[R_NEXT+SIZE_A]	LINK_B[R_NEXT]
1	NULL	5	1	1	6+1	3	2
2	5	1	2	0	6+1	3	2
3	1	0	3	2	6+1	3	2
4	0	2	5	4	6+1	3	2
5	7	2	5	4	6+2	4	0
6	8	2	5	4	6+0	8	-1
7	2	4	6	3	6+0	8	-1
8	4	3	7	-1	6+0	8	-1
9	3	-1					
10					-1		

HEAD_A =5
HEAD_B =1
HEAD_C =5

ITERATION		LINK_C	0	1	2	3	4	5	6	7	8
1	HEAD_C =5	LINK_C									
2		LINK_C						1			
3		LINK_C		0							
4		LINK_C	7								
5		LINK_C								8	
6		LINK_C									2
7		LINK_C			4						
8		LINK_C					3				
9		LINK_C				6					
10		LINK_C							-1		
11		LINK_C	7	0	4	6	3	1	-1	8	2

```
/**
@file

A program that concatenates two arrays in one array and merge their linked lists in one linked lists asceding order without using any search ir

@author

Name             :      Muhammed Yasin SAGLAM
Student No       :      15011804
Date             :      08/03/2017
E-Mail           :      myasinsaglam1907@gmail.com
Compiler Used    :      GCC
IDE              :      DEV-C++(Version 5.11)
Operating System :      Windows 10 educational edition
*/
#include<stdio.h>
#include<stdlib.h>

//Function procedures that used in this program
void print_arr(int *arr,int size);
void concatenate_arr(int *arr1,int *arr2,int *arr3,int size1,int size2);
void merge_link(int *arr,int *link1,int head1,int *link2,int head2,int *link3,int *head3,int SIZE_A);
int *Allocator(int size);
void display_sorted(int *arr_C,int *link_C,int head_C);

/**
Main function, Reads array size from the user.Then it makes allocation and value assignment.
After it concatenate A array and B array(in C array) it creates link array merging them as a ascending links(link C array).
Then it prints outputs on the screen(Head C, Link C array and Array C).
*/
int main(){

int *arr_A,*arr_B,*arr_C; // Array declarations (arr_A is A array, arr_B is B array, arr_C is C array(concatenation of A and B arrays) )
int *link_A,*link_B,*link_C; // Link Array declarations (link_A is Links of A array, link_B is Links of B array, link_C is links of C array)
int head_A,head_B,head_C; // Head Variables for A,B and C linked arrays
int SIZE_A,SIZE_B,SIZE_C; // Size Variables of arrays that related with A,B and C
int i; // Variable for iterations
//TAKING ALL VALUES OF ARRAYS AND LINK ARRAYS FROM USER WITH HEAD VALUES...
//INPUTS FOR ARRAY A
    printf("Enter size of array_A : ");
    scanf("%d",&SIZE_A);                                        //Reading size of A array from user.
    arr_A=Allocator(SIZE_A);                                    //Memory allocation for A labeled arrays by calling Allocator funct
    link_A=Allocator(SIZE_A);

    printf("\nEnter the all(%d) elements of array_A : ",SIZE_A);
    for(i=0;i<SIZE_A;i++){
    //  printf("\n%d. element : ",i+1);                           //print for step by step reading from user. Not necessary.
        scanf("%d",&arr_A[i]);                                  //Reading the values of A labeled array from user.
    }
    system("CLS");
    printf("Array_A created...\n\nEnter link array A !!!\nArray:");
    print_arr(arr_A,SIZE_A);                                    //Printing taken array values on the screen by calling print_arr fu
    printf("\nHEAD A: ");
    scanf("%d",&head_A);                                        //Reading head value of A labeled link array.
    printf("\n");
    printf("LINK A :");
    for(i=0;i<SIZE_A;i++){
        scanf("%d",&link_A[i]);                                 //Reading the contents of A labeled link array.
    }
    printf("\nLink_A array created successfuly\n");

    printf("ARRAY A : ");
    print_arr(arr_A,SIZE_A);                                    //Printing A labeled value array on the screen by calling function.
    printf("LINK  A : ");
    print_arr(link_A,SIZE_A);                                   //Printing A labeled link array on the screen by calling function.
    system("PAUSE");
    system("CLS");

    //INPUTS FOR ARRAY B
    printf("Enter size of array_B : ");
    scanf("%d",&SIZE_B);                                        //Reading size of B array from user.
    arr_B=Allocator(SIZE_B);                                    //Memory allocation for B labeled arrays by calling Allocator funct
    link_B=Allocator(SIZE_B);

    printf("Please enter the all(%d) elements of array_B : ",SIZE_B);
    for(i=0;i<SIZE_B;i++){
    //  printf("\n%d. element : ",i+1);                           //print for step by step reading from user. Not necessary.
        scanf("%d",&arr_B[i]);                                  //Reading the values of B labeled array from user.
    }
    system("CLS");
    printf("Array B created...\n\nEnter link array B !!!\nArray:");
    print_arr(arr_B,SIZE_B);                                    //Printing taken array values on the screen by calling print_arr fu
    printf("\nHEAD B: ");
    scanf("%d",&head_B);                                        //Reading head value of B labeled link array.
    printf("\n");
    printf("LINK B :");
    for(i=0;i<SIZE_B;i++){
        scanf("%d",&link_B[i]);                                 //Reading the contents of B labeled link array.
    }
    printf("\nLinkB array created successfuly\n");

    printf("ARRAY B : ");
    print_arr(arr_B,SIZE_B);                                    //Printing B labeled value array on the screen by calling function.
    printf("LINK  B : ");
    print_arr(link_B,SIZE_B);                                   //Printing A labeled link array on the screen by calling function.
    system("PAUSE");
```

```
    system("CLS");

    SIZE_C=SIZE_A+SIZE_B;                                                   //Assingning the Size value of a C labeled array.
    arr_C=Allocator(SIZE_C);                                                //Memory allocation for C labeled arrays by calling Allocator funct
    link_C=Allocator(SIZE_C);

    concatenate_arr(arr_A,arr_B,arr_C,SIZE_A,SIZE_B);                       //Array concatenation(A+B=C) operations by calling concatenate_arr
    free(arr_A);                                                            //Free A labeled value array. We dont need anymore.
    free(arr_B);                                                            //Free B labeled value array. We dont need anymore.

    printf("\nConcatenation completed succesfully...\n");
    printf("ARRAY C :");
    print_arr(arr_C,SIZE_C);                                                //Printing C labeled new array on the screen.

    merge_link(arr_C,link_A,head_A,link_B,head_B,link_C,&head_C,SIZE_A);     //Assigning values of link C array as a ascending order by calling

    printf("\nLINK  C :");
    print_arr(link_C,SIZE_C);                                               //Print link C array on the screen to see the results of algorithm.
    printf("\nHEAD C: %d\n",head_C);                                        //Print head value of a C labeled link array on the screen to see t

    display_sorted(arr_C,link_C,head_C);                                    //Prints sorted array as a result of algorithm on the screen.

//Freeing operations of resting allocated arrays.
free(arr_C);
free(link_C);
free(link_B);
free(link_A);

system("PAUSE");
return 0;
}
/**
    @param *arr     pointer of an array that will be print on the screen.
    @param size     the size value of an array that will be print on the screen.
*/
void print_arr(int *arr,int size){
    int i;              //iteration variable
    for(i=0;i<size;i++)
        printf(" %-3d ",arr[i]);    //Printing operation.
    printf("\n");
}

/**
    @param *arr1    the pointer of a first source array of concatenation operation
    @param *arr2    the pointer of a second source array of concatenation operation
    @param *arr3    the pointer of a destination array of concatenation operation
    @param size1    the size of a first source array of concatenation operation
    @param size2    the size of a second source array of concatenation operation
*/
void concatenate_arr(int *arr1,int *arr2,int *arr3,int size1,int size2){
    int i; //iteration variable
    for(i=0;i<size1;i++)
        arr3[i]=arr1[i]; //copying first source array values to destination array.
    for(i=size1;i<size1+size2;i++)
        arr3[i]=arr2[i-size1]; //copying second source array values to destination array.
}

/**
    @param  *arr        the pointer of a destination array of concatenation operation. (C labeled value array in main)
    @param  *link1      the pointer of a first ordered link array as a source.  (A labeled link array in main)
    @param  head1       the  head value of a first ordered link array.
    @param  *link2      the pointer of a second ordered link array as a source. (B labeled link array in main)
    @param  head2       the head value of a second ordered link array.
    @param  *link3      the pointer of a destination
    @param  *head3      the pointer of new link array's head value.(Head of C labeled link array.)
    @param  SIZE_A      the size of a first ordered link array as a source. (size of A labeled link array in main).
*/
void merge_link(int *arr,int *link1,int head1,int *link2,int head2,int *link3,int *head3,int SIZE_A){
    int l_next; //iteration variable of first source array(link_A array)
    int r_next; //iteration variable of second source array(link_B array)
    int cur;    //iteration variable of destination array(link_C array)
    //initial(head) value assigment to iteration variables of arrays. No need but good for code readability and clearity
    l_next=head1;
    r_next=head2;
    //control to determine the destination array's(link_C) head value(head_C)
    if(arr[head1]<=arr[head2+SIZE_A]){
        *head3=head1; //head_C assignment
        l_next=link1[head1]; //forward next link of first array
        cur=head1; //assign  first array's head index to current position of link_C array (starting point)
    }
    else{
        *head3=head2+SIZE_A; //head_C assignment
        r_next=link2[head2]; //forward next link of second array
        cur=head2+SIZE_A;   //assign second array's head index to current position of link_C array (starting point)
    }

    while(l_next!=-1 && r_next!=-1){ // we can understand the end of each array by controlling equals to -1 value or not
        if(arr[l_next] <= arr[r_next+SIZE_A]){
            link3[cur]=l_next;  //link_C value assignment
            cur=l_next;         //Change current index on link_C array for coming next indexes
            l_next=link1[l_next]; //forward next link of first array
        }
        else{
            link3[cur]=r_next+SIZE_A;   //link_C value assignment
            cur=SIZE_A+r_next;          //Change current index on link_C array for coming next indexes
            r_next=link2[r_next];       //forward next link of second array
```

```
        }
    }

    while(l_next!=-1){ //if the first array has still elements that not assigned. The Assignment of link_C value operations continues on link_A
        link3[cur]=l_next;     //link_C value assignment
        cur=l_next;            //Change current index on link_C array for coming next indexes
        l_next=link1[l_next];  //forward next link of first array
    }

    while(r_next!=-1){ //if the second array has still elements that not assigned. The Assignment of link_C value operations continues on link_
        link3[cur]=r_next+SIZE_A;   //link_C value assignment
        cur=SIZE_A+r_next;          //Change current index on link_C array for coming next indexes
        r_next=link2[r_next];       //forward next link of second array
    }

    link3[cur]=-1;                  //after all we assign last value as -1 to finish link_C array value assignment.
}
/**
    @param size         size of array that will be allocate
    @return             the address of an array that allocated
*/
int *Allocator(int size){
    int *array;
    array=(int*)malloc(sizeof(int)*size);
        if(!array){
            system("COLOR c");
            printf("Array Not Allocated !!! Quitting...");
            exit(0);
        }
    return array;
}

/**
    @param *arr_C       the array that contains not ordered numbers.
    @param *link_C      the link array of arr_C array.
    @param head_C       the head value of a link_C array.
*/
void display_sorted(int *arr_C,int *link_C,int head_C){
    //initial(head) value assigment to iteration variable of an array. No need but good for code readability and clearity
    int next;
    next=head_C;
    printf("\nSORTED C ARRAY ARGUMENTS ARE :");
    while(next!=-1){
        printf(" %-3d ",arr_C[next]); //prints value of array by following order of link array
        next=link_C[next]; //forward next linked argument of an array
    }
    printf("\n");
}
```