Ankan Basu MTCT Sem 1 002310504015 (Programming Lab Assignment 1)

Write a menu driven program in C to perform the following operations in an integer array allocated memory dynamically. The list may grow or shrink compared to the initial allocation as and when required. Multiple operations may be performed on the existing list without recompiling/re executing the program. The operations are:
a) Create, **b)** Count, **c)** Indexed element, **d)** Insert, **e)** Delete, **f)** Merge, **g)** Split, **h)** Sort, **j)** Search

PROGRAM

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

int *create_array(int *size);
int *insert(int *arr, int *size, int indx, int element);
int *delete_element(int *arr, int *size, int element);
int search(int *arr, int size, int element);
int *merge(int *arr1, int *arr2, int size1, int size2, int
*size3);
int *split(int *arr, int *arr2, int *size, int *size2, int indx);
void sort(int *arr, int start, int end);
int partition(int *arr, int start, int end);
void swap(int *a, int *b);
void print_array(int *arr, int size);

int main() {
   int n;
   int *arr=NULL, size, indx, element, flg=1, *arr2=NULL, size2,
*arr3=NULL, size3;
   while(flg) {
      printf("\n1. Create\n");
   printf("2. Count\n");
   printf("3. Indexed Element\n");
   printf("4. Insert\n");
   printf("5. Delete\n");
   printf("6. Merge\n");
   printf("7. Split\n");
   printf("8. Sort\n");
   printf("9. Search\n");
   printf("0. Exit\n");
   printf("Chose option: ");
   scanf("%d", &n);
```

```c
switch(n) {
   case 0:
      flg=0;
      break;
   case 1:
      if (arr != NULL) {//free previously created array, if any
         free(arr);
         arr = NULL;
         size = 0;
      }
      arr = create_array(&size);
      fflush(stdin);
      print_array(arr, size);
      break;
   case 2:
      printf("Size of array: %d\n", size);
      break;
   case 3:
      if (arr == NULL) {
         printf("Create array first\n");
         break;
      }
      printf("Enter index: ");
      scanf("%d", &indx);
      if (indx >= size) {
         printf("Array Index Out of Bounds.\n");
         break;
      }
      printf("Element: %d\n", arr[indx]);
      break;
   case 4:
      if (arr == NULL) {
         printf("Create array first\n");
         break;
      }
      printf("Enter index to insert: ");
      scanf("%d", &indx);

      if (indx > size) {
         printf("Array Index Out of Bounds.\n");
         break;
      }
      printf("Enter element to insert: ");
      scanf("%d", &element);
```

```c
            arr = insert(arr, &size, indx, element);
            print_array(arr, size);
            break;
         case 5:
            if (arr == NULL) {
               printf("Create array first\n");
               break;
            }
            printf("Enter element to delete: ");
            scanf("%d", &element);
            arr = delete_element(arr, &size, element);
            print_array(arr, size);
            break;
         case 6:
            if (arr == NULL) {
               printf("Create array 1\n");
               arr = create_array(&size);
            }
            if (arr2 != NULL) { // free previously created array
               free(arr2);
               arr2 = NULL;
               size2 = 0;
            }
            printf("Create Array2:\n");
            arr2 = create_array(&size2);
            if (arr3 != NULL) { // free previously created array
               free(arr3);
               arr3 = NULL;
               size3 = 0;
            }
            arr3 = merge(arr, arr2, size, size2, &size3);
            print_array(arr3, size3);
            break;
         case 7:
            if (arr == NULL) {
               printf("Create array first\n");
               break;
            }
            printf("Enter index to split: ");
            scanf("%d", &indx);
            if (indx > size) {
               printf("Array Index Out of Bounds.\n");
               break;
```

```c
          }
          if (arr2 != NULL) { // free previously created array
             free(arr2);
             arr2 = NULL;
             size2 = 0;
          }
          arr2 = split(arr, arr2, &size, &size2, indx);
          print_array(arr, size);
          print_array(arr2, size2);
          break;
       case 8:
          if (arr == NULL) {
             printf("Create array first\n");
             break;
          }
          sort(arr, 0, size-1);
          print_array(arr, size);
          break;
       case 9:
          if (arr == NULL) {
             printf("Create array first\n");
             break;
          }
          printf("Enter Element: ");
          scanf("%d", &element);
          indx = search(arr, size, element);
          printf("Index: %d", indx);
          break;
       default:
          printf("Invalid Input\n");
          break;
       }
    }

    // free the memory
    if (arr != NULL) {
       free(arr);
       arr = NULL;
       size = 0;
    }
    if (arr2 != NULL) {
       free(arr2);
       arr2 = NULL;
       size2 = 0;
```

```c
   }
   if (arr3 != NULL) {
      free(arr3);
      arr3 = NULL;
      size3 = 0;
   }
   return 0;
}

int *create_array(int *size) {
   // takes comma seperated numbers as input and creates array
   int *arr = NULL;
   int x, y, z, flg=1;
   (*size) = 0;

   printf("Enter array elements: ");
   while(flg) {
      z = scanf("%d", &x);
      y = getchar();
      if (y == '\n') { // end of input
         flg=0;
      }
      if (z == 0 && (y!='\n' || y !=',' || y != ' ')) {
// if anything other than numbers or new line or space entered
         printf("Invalid Input\n");
         break;
      }
      (*size)++;
      arr = realloc(arr, (*size)*sizeof(int)); // increment size
of array
      if (arr == NULL) {
         printf("MEMORY ALLOCATION ERROR.\n");
         exit(-1);
      }
      arr[(*size)-1] = x; // insert element at end.
   }
   return arr;
}

int *delete_element(int *arr, int *size, int element) {
   // search for element
   int indx = -1;
   indx = search(arr, *size, element);
   if (indx == -1) {
      printf("Element %d not found\n", element);
      return arr;
   }

   // shift elements backwards
   for (int i=indx+1; i<*size; i++) {
      arr[i-1] = arr[i];
   }
   (*size)--;
   arr = realloc(arr, (*size)*(sizeof(int)));

   return arr;
}

int search(int *arr, int size, int element) {
   int indx = -1;
   for (int i=0; i<size; i++) {
      if (arr[i] == element) {
         indx = i;
         break;
      }
   }
   return indx;
}

int *merge(int *arr1, int *arr2, int size1, int size2, int *size3)
{
   int *arr3 = NULL;
   *size3 = size1+size2; // size of array after merge

   arr3 = malloc((*size3)*sizeof(int));
   if (arr3 == NULL) {
      printf("MEMORY ALLOCATION ERROR\n");
      exit(-1);
   }

   for (int i=0; i<size1; i++) {
      arr3[i] = arr1[i];
   }
   for (int i=0; i<size2; i++) {
      arr3[size1+i] = arr2[i];
   }

   return arr3;
```

```
}

int *split(int *arr, int *arr2, int *size, int *size2, int indx) {
   int new_size = *size - indx - 1; // size of new array
   *size2 = new_size;
   *size = indx+1; // size of the other array after split

   arr2 = realloc(arr2, (*size2)*(sizeof(int)));
   if (arr2 == NULL) {
      printf("MEMORY ALLOCATION ERROR\n");
      exit(-1);
   }
   for (int i=0; i<(*size2); i++) {
      arr2[i] = arr[(*size)+i]; // moving elem to new array after
split
   }
   arr = realloc(arr, (*size)*(sizeof(int)));
   if (arr == NULL) {
      printf("MEMORY ALLOCATION ERROR\n");
      exit(-1);
   }

   return arr2;
}

int *insert(int *arr, int *size, int indx, int element) {
   (*size)++;
   arr = realloc(arr, (*size)*sizeof(int));
   if (arr == NULL) {
      printf("MEMORY ALLOCATION ERROR\n");
      exit(-1);
   }

   // move elements to the right;
   for (int i=(*size)-1; i>indx; i--) {
      arr[i] = arr[i-1];
   }
   arr[indx] = element;

   return arr;
}

void sort(int *arr, int start, int end) {
   // Quick sort algorithm
```

```c
   int pivot_indx;
   if (start < end) {
      pivot_indx = partition(arr, start, end);
      sort(arr, start, pivot_indx-1);
      sort(arr, pivot_indx+1, end);
   } else {
      return;
   }
}

void swap(int *a, int *b) {
   // function to swap 2 variables
   int tmp;
   tmp = *a;
   *a = *b;
   *b = tmp;
}

int partition(int *arr, int start, int end) {
   // Lomuto partition for quicksort
   int pivot, i, tmp;
   pivot = arr[end];

   i = start-1;
   for(int j = start; j<= end-1; j++) {
      if (arr[j] <= pivot) {
         i = i + 1;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i+1], &arr[end]);
   return i+1;
}

void print_array(int *arr, int size) {
   printf("{");
   for (int i=0; i<size; i++) {
      printf("%d", arr[i]);
      if (i != size-1) {
         printf(", ");
      }
   }
   printf("}\n");
}
```

OUTPUT

```
1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 1
Enter array elements: 1,5,3,2,-3
{1, 5, 3, 2, -3}

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 2
Size of array: 5

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 3
Enter index: 1
Element: 5

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
```

```
9. Search
0. Exit
Chose option: 4
Enter index to insert: 1
Enter element to insert: 0
{1, 0, 5, 3, 2, -3}

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 5
Enter element to delete: 2
{1, 0, 5, 3, -3}

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 8
{-3, 0, 1, 3, 5}

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 1
Enter array elements: 1,3,4,2,7,8,9,0,4,2
{1, 3, 4, 2, 7, 8, 9, 0, 4, 2}

1. Create
2. Count
3. Indexed Element
4. Insert
```

```
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 7
Enter index to split: 3
{1, 3, 4, 2}
{7, 8, 9, 0, 4, 2}

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 1
Enter array elements: 1,2,3,4
{1, 2, 3, 4}

1. Create
2. Count
3. Indexed Element
4. Insert
5. Delete
6. Merge
7. Split
8. Sort
9. Search
0. Exit
Chose option: 6
Create Array2:
Enter array elements: 4,5,8
{1, 2, 3, 4, 4, 5, 8}
```