|  |  |
| --- | --- |
| Data Structure  [2022] |  |
|  |  |
| [01.09.2022]  Techno India College of Technology  Authored by: Abu Taha Md Farhad  Roll: 31401221052  Reg No: 213141001210016 OF 2021-22  Stream: BCA 2nd year  Subject: Data Structure Lab [BCAC393] |  |

Code:

# Insertion:

## array\_insert\_beginning.c:

*int* insert\_at\_beginning(*int* \**arr*, *int*\* *n*, *int* *val*){

    for (*int* i = \**n*; i >= 1; i--)

    {

        \*(*arr*+i) = \*(*arr*+(i-1));

    }

    \**arr* = *val*;

    \**n* = (\**n*)+1;

}

## array\_insert\_last.c:

*int* insert\_at\_last(*int* \**arr*, *int*\* *n*, *int* *val*){

    \*(*arr*+\**n*) = *val*;

    \**n* = (\**n*)+1;

}

## array\_insert\_position.c:

*int* insert\_at\_position(*int* \**arr*, *int*\* *n*, *int* *val*, *int* *pos*){

    for (*int* i = \**n*; i >= *pos*; i--)

    {

        \*(*arr*+i) = \*(*arr*+(i-1));

    }

    \*(*arr*+*pos*) = *val*;

    \**n* = (\**n*)+1;

}

# Deleteion:

## array\_delete\_beginning.c:

*int* delete\_from\_beginning(*int* \**arr*, *int*\* *n*) {

    for (*int* i = 0; i < \**n*-1; i++)

    {

        \*(*arr*+i) = \*(*arr*+(i+1));

    }

    \*(*arr*+(\**n*-1)) = 0;

    \**n* = (\**n*)-1;

}

## array\_delete\_last.c

*int* delete\_from\_last(*int* \**arr*, *int*\* *n*){

    \*(*arr*+(\**n*-1)) = 0;

    \**n* = (\**n*)-1;

}

## array\_delete\_position.c:

*int* delete\_from\_position(*int* \**arr*, *int*\* *n*, *int* *pos*){

    for (*int* i = *pos*; i < \**n*-1; i++)

    {

        \*(*arr*+i) = \*(*arr*+(i+1));

    }

    \*(*arr*+(\**n*-1)) = 0;

    \**n* = (\**n*)-1;

}

# Searching:

## array\_search\_linear.c:

*int* search\_linear(*int* \**arr*, *int* *n*, *int* *search\_val*) {

    for (*int* i = 0; i < *n*; i++)

    {

        if (\*(*arr*+i) == *search\_val*)

        {

            return i;

        }

    }

    return -1;

}

## array\_search\_binary.c:

*int* search\_binary(*int*\* *arr*, *int* *n*, *int* *search\_val*){

*int* beg = 0;

*int* end = *n*-1;

    while (beg <= end)

    {

*int* mid = (beg + end) / 2;

        if (\*(*arr*+mid) == *search\_val*)

        {

            return mid;

        }

        else if (\*(*arr*+mid) > *search\_val*)

        {

            end = mid - 1;

        }

        else

        {

            beg = mid + 1;

        }

    }

    return -1;

}

# Sorting:

## array\_sort\_bubble.c:

*void* sort\_bubble(*int* \**arr*, *int* *n*) {

  for (*int* i = 0; i < *n*; i++)

  {

    for (*int* j = i + 1; j < *n*; j++)

    {

      if (\*(*arr*+i) > \*(*arr*+j))

      {

*int* temp = \*(*arr*+i);

        \*(*arr*+i) = \*(*arr*+j);

        \*(*arr*+j) = temp;

      }

    }

  }

}

## array\_sort\_selection.c:

*void* sort\_selection(*int* \**arr*, *int* *n*) {

  for (*int* i = *n* - 1; i >= 1; i--)

  {

*int* max = \**arr*;

*int* index = 0;

    for (*int* j = 1; j <= i; j++)

    {

      if (\*(*arr*+j) > max)

      {

        max = \*(*arr*+j);

        index = j;

      }

    }

    \*(*arr*+index) = \*(*arr*+i);

    \*(*arr*+i) = max;

  }

}

## array\_sort\_insertion.c:

*void* sort\_insertion(*int* \**arr*, *int* *n*)

{

    for (*int* i = 1; i < *n*; i++)

    {

*int* temp = \*(*arr* + i);

        for (*int* j = i - 1; j >= 0; j--)

        {

            if (\*(*arr* + j) > temp)

            {

                \*(*arr* + (j + 1)) = \*(*arr* + j);

                \*(*arr* + j) = temp;

            }

            else

            {

                break;

            }

        }

    }

}

## array\_sort\_merge.c:

#include <malloc.h>

*int* sizeOfOriginalArray;

*void* merge(*int* \*, *int*, *int*, *int*);

*void* merge\_sort(*int* \*, *int*, *int*);

*void* merge(*int*\* *arr*, *int* *beg*, *int* *mid*, *int* *end*)

{

*int* i = *beg*, j = *mid* + 1, index = *beg*, k;

*int*\* temp = (*int* \*)calloc(sizeOfOriginalArray, sizeof(*int*));

    while ((i <= *mid*) && (j <= *end*))

    {

        if (\*(*arr* + i) < \*(*arr* + j))

        {

            \*(temp + index) = \*(*arr* + i);

            i++;

        }

        else

        {

            \*(temp + index) = \*(*arr* + j);

            j++;

        }

        index++;

    }

    if (i > *mid*)

    {

        while (j <= *end*)

        {

            \*(temp + index) = \*(*arr* + j);

            j++;

            index++;

        }

    }

    else

    {

        while (i <= *mid*)

        {

            \*(temp + index) = \*(*arr* + i);

            i++;

            index++;

        }

    }

    for (k = *beg*; k < index; k++)

        \*(*arr* + k) = \*(temp + k);

}

*void* merge\_sort(*int*\* *arr*, *int* *beg*, *int* *end*)

{

*int* mid;

    if (*beg* < *end*)

    {

        mid = (*beg* + *end*) / 2;

        merge\_sort(*arr*, *beg*, mid);

        merge\_sort(*arr*, mid + 1, *end*);

        merge(*arr*, *beg*, mid, *end*);

    }

}

*void* sort\_merge(*int* \**arr*, *int* *n*) {

    sizeOfOriginalArray = *n*;

    // main merge sort function call

    merge\_sort(*arr*, 0, *n*-1);

}

## array\_sort\_quick.c:

*void* quicksort(*int*\*, *int*, *int*);

*int* partition(*int*\*, *int*, *int*);

*void* quicksort(*int* \**a*, *int* *beg*, *int* *end*)

{

*int* loc;

    if (*beg* < *end*)

    {

        loc = partition(*a*, *beg*, *end*);

        quicksort(*a*, *beg*, loc - 1);

        quicksort(*a*, loc + 1, *end*);

    }

}

*int* partition(*int* \**a*, *int* *beg*, *int* *end*)

{

*int* left, right, loc, temp, flag = 0;

    left = loc = *beg*;

    right = *end*;

    while (flag != 1)

    {

        while (\*(*a* + loc) <= \*(*a* + right) && loc != right)

            right--;

        if (loc == right)

            flag = 1;

        else if (\*(*a* + loc) > \*(*a* + right))

        {

            temp = \*(*a* + loc);

            \*(*a* + loc) = \*(*a* + right);

            \*(*a* + right) = temp;

            loc = right;

        }

        if (flag != 1)

        {

            while (\*(*a* + loc) >= \*(*a* + left) && loc != left)

                left++;

            if (loc == left)

                flag = 1;

            else if (\*(*a* + loc) < \*(*a* + left))

            {

                temp = \*(*a* + loc);

                \*(*a* + loc) = \*(*a* + left);

                \*(*a* + left) = temp;

                loc = left;

            }

        }

    }

    return loc;

}

*void* sort\_quick(*int*\* *arr*, *int* *n*){

    quicksort(*arr*, 0, *n* - 1);

}

# Other setup files:

## array\_insert\_many.c:

*void* insert\_many(*int* \**arr*, *int* \**n*)

{

*int* noOfElements, val;

    printf("Enter the number of elements: ");

    scanf("%d", &noOfElements);

    for (*int* i = 0; i < noOfElements; i++)

    {

        printf("array[%d] = ", (\**n*));

        scanf("%d", &val);

        insert\_at\_last(*arr*, *n*, val);

    }

}

## array\_display\_by\_value.c:

*void* display\_array(*int* *arr*[], *int* *count*){

    for (*int* i = 0; i < *count*; i++)

    {

        printf("%d", *arr*[i]);

        if (i < *count* - 1)

        {

            printf(", ");

        }

    }

}

## array\_case.c:

*void* help()

{

    printf("0 - Help\n");

    printf("1 - Insert\n");

    printf("\t1 - Beginning\n");

    printf("\t2 - Middle\n");

    printf("\t3 - Last\n");

    printf("2 - Delete\n");

    printf("\t1 - Beginning\n");

    printf("\t2 - Middle\n");

    printf("\t3 - Last\n");

    printf("3 - Search\n");

    printf("\t1 - Linear\n");

    printf("\t2 - Binary\n");

    printf("4 - Sort\n");

    printf("\t1 - Bubble\n");

    printf("\t2 - Insertion\n");

    printf("\t3 - Selection\n");

    printf("\t4 - Merge\n");

    printf("\t5 - Quick\n");

    printf("5 - Display array\n");

    printf("6 - Insert many\n");

    printf("7 - Quit\n");

}

*void* insert(*int* \**arr*, *int* \**n*)

{

*int* option, val, pos;

    printf("Enter the type of insertion: ");

    scanf("%d", &option);

    printf("Enter the value to be inserted: ");

    scanf("%d", &val);

    switch (option)

    {

    case 0:

        help();

        break;

    case 1:

        insert\_at\_beginning(*arr*, *n*, val);

        break;

    case 2:

        printf("Enter the position: ");

        scanf("%d", &pos);

        insert\_at\_position(*arr*, *n*, val, pos);

        break;

    case 3:

        insert\_at\_last(*arr*, *n*, val);

        break;

    default:

        printf("Enter a valid command\n");

        help();

        break;

    }

}

*void* delete (*int* \**arr*, *int* \**n*)

{

*int* option, pos;

    printf("Enter the type of deletion: ");

    scanf("%d", &option);

    switch (option)

    {

    case 0:

        help();

        break;

    case 1:

        delete\_from\_beginning(*arr*, *n*);

        break;

    case 2:

        printf("Enter the position: ");

        scanf("%d", &pos);

        delete\_from\_position(*arr*, *n*, pos);

        break;

    case 3:

        delete\_from\_last(*arr*, *n*);

        break;

    default:

        printf("Enter a valid command\n");

        help();

        break;

    }

}

*void* search(*int* \**arr*, *int* *n*)

{

*int* option, val, pos;

    printf("Enter the type of search: ");

    scanf("%d", &option);

    printf("Enter the value to be searched: ");

    scanf("%d", &val);

    switch (option)

    {

    case 0:

        help();

        break;

    case 1:

        pos = search\_linear(*arr*, *n*, val);

        break;

    case 2:

        pos = search\_binary(*arr*, *n*, val);

        break;

    default:

        printf("Enter a valid command\n");

        help();

        break;

    }

    if (pos > -1)

    {

        printf("a[%d] = %d\n", pos, val);

    }

    else

    {

        printf("Element not found.\n");

    }

}

*void* sort(*int* \**arr*, *int* *n*)

{

*int* option;

    printf("Enter the type of sort: ");

    scanf("%d", &option);

    switch (option)

    {

    case 0:

        help();

        break;

    case 1:

        sort\_bubble(*arr*, *n*);

        break;

    case 2:

        sort\_insertion(*arr*, *n*);

        break;

    case 3:

        sort\_selection(*arr*, *n*);

        break;

    case 4:

        sort\_merge(*arr*, *n*);

        break;

    case 5:

        sort\_quick(*arr*, *n*);

        break;

    default:

        printf("Enter a valid command\n");

        help();

        break;

    }

}

## array\_utilities.h:

#include "array\_display\_by\_value.c"

#include "array\_insert\_beginning.c"

#include "array\_insert\_position.c"

#include "array\_insert\_last.c"

#include "array\_insert\_many.c"

#include "array\_delete\_beginning.c"

#include "array\_delete\_position.c"

#include "array\_delete\_last.c"

#include "array\_search\_linear.c"

#include "array\_search\_binary.c"

#include "array\_sort\_bubble.c"

#include "array\_sort\_selection.c"

#include "array\_sort\_insertion.c"

#include "array\_sort\_merge.c"

#include "array\_sort\_quick.c"

#include "array\_case.c"

# Main File:

## main.c:

#include <stdio.h>

#include "array\_utilities.h"

*int* main()

{

    // number of elements initialized in array arr[]

    // must be equal to the value of variable n

*int* arr[] = {};

*int* n = 0;

*int* option = 0;

    while (1)

    {

        switch (option)

        {

        case 0:

            help();

            break;

        case 1:

            insert(arr, &n);

            break;

        case 2:

            delete (arr, &n);

            break;

        case 3:

            search(arr, n);

            break;

        case 4:

            sort(arr, n);

            break;

        case 5:

            printf("array[%d] = {", n);

            display\_array(arr, n);

            printf("}\n");

            break;

        case 6:

            insert\_many(arr, &n);

            break;

        case 7:

            return 0;

            break;

        default:

            printf("Enter a valid command\n");

            help();

            break;

        }

        printf("> ");

        scanf("%d", &option);

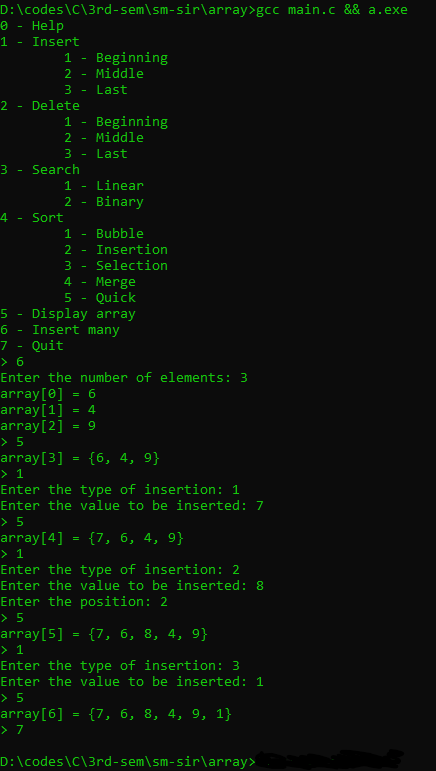
    }

    return 0;

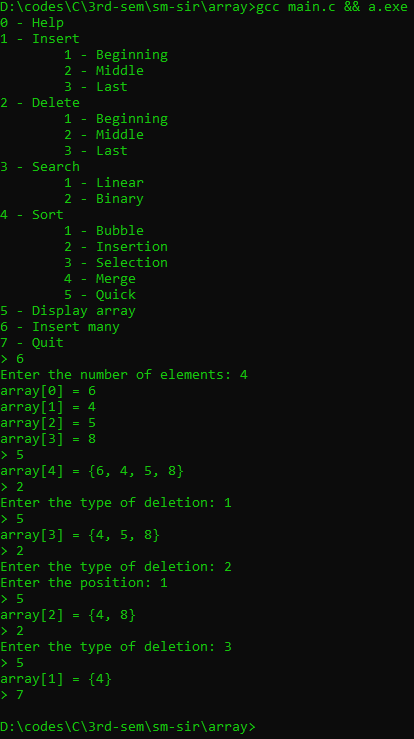
}

Output:

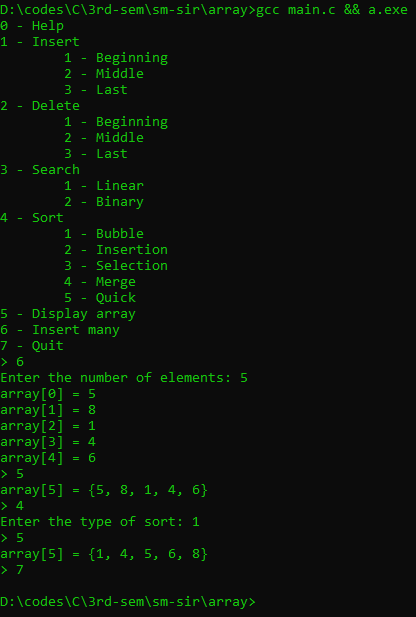
# Insertion:



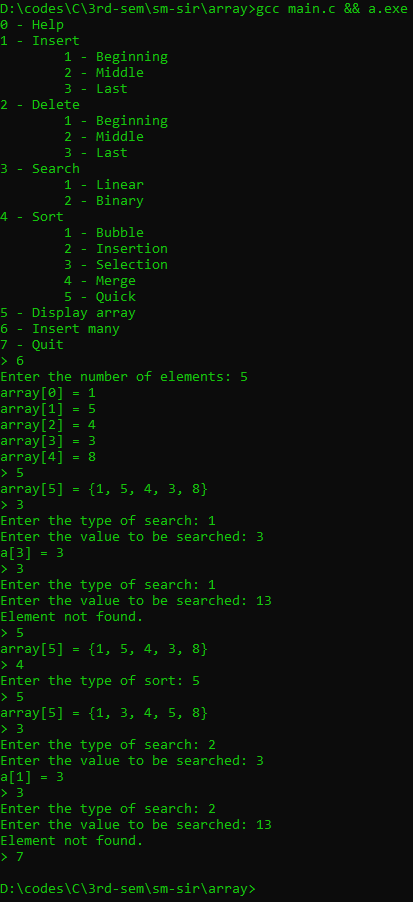
# Deletion:



# Sort:



# Search:



# repo

## Thank You