

```

1: //Array Menu using C
2:
3: #include <stdio.h>
4: #include<stdlib.h>
5:
6: struct Array
7: {
8:     int *A;
9:     int size;
10:    int length;
11: };
12:
13: void Display(struct Array arr)
14: {
15:     int i;
16:     printf("\nElements are\n");
17:     for(i=0;i<arr.length;i++)
18:         printf("%d ",arr.A[i]);
19: }
20:
21: void Append(struct Array *arr,int x)
22: {
23:     if(arr->length<arr->size)
24:         arr->A[arr->length++]=x;
25: }
26:
27: void Insert(struct Array *arr,int index,int x)
28: {
29:     int i;
30:     if(index>=0 && index <=arr->length)
31:     {
32:         for(i=arr->length;i>index;i--)
33:             arr->A[i]=arr->A[i-1];
34:         arr->A[index]=x;
35:         arr->length++;
36:     }
37: }
38:
39: int Delete(struct Array *arr,int index)

```

```

40: {
41:     int x=0;
42:     int i;
43:
44:     if(index>=0 && index<arr->length)
45:     {
46:         x=arr->A[index];
47:         for(i=index;i<arr->length-1;i++)
48:             arr->A[i]=arr->A[i+1];
49:         arr->length--;
50:         return x;
51:     }
52:     return 0;
53: }
54:
55: void swap(int *x,int *y)
56: {
57:     int temp;
58:     temp=*x;
59:     *x=*y;
60:     *y=temp;
61: }
62:
63: int LinearSearch(struct Array *arr,int key)
64: {
65:     int i;
66:     for(i=0;i<arr->length;i++)
67:     {
68:         if(key==arr->A[i])
69:         {
70:             swap(&arr->A[i],&arr->A[0]);
71:             return i;
72:         }
73:     }
74:     return -1;
75: }
76:
77: int BinarySearch(struct Array arr,int key)
78: {

```

```

79:  int l,mid,h;
80:  l=0;
81:  h=arr.length-1;
82:
83:  while(l<=h)
84:  {
85:      mid=(l+h)/2;
86:      if(key==arr.A[mid])
87:          return mid;
88:      else if(key<arr.A[mid])
89:          h=mid-1;
90:      else
91:          l=mid+1;
92:  }
93:  return -1;
94: }
95:
96: int RBinSearch(int a[],int l,int h,int key)
97: {
98:     int mid;
99:
100:    if(l<=h)
101:    {
102:        mid=(l+h)/2;
103:        if(key==a[mid])
104:            return mid;
105:        else if(key<a[mid])
106:            return RBinSearch(a,l,mid-1,key);
107:        else
108:            return RBinSearch(a,mid+1,h,key);
109:    }
110:    return -1;
111: }
112:
113: int Get(struct Array arr,int index)
114: {
115:     if(index>=0 && index<arr.length)
116:         return arr.A[index];
117:     return -1;

```

```
118: }
119:
120: void Set(struct Array *arr,int index,int x)
121: {
122:     if(index>=0 && index<arr->length)
123:         arr->A[index]=x;
124: }
125:
126: int Max(struct Array arr)
127: {
128:     int max=arr.A[0];
129:     int i;
130:     for(i=1;i<arr.length;i++)
131:     {
132:         if(arr.A[i]>max)
133:             max=arr.A[i];
134:     }
135:     return max;
136: }
137:
138: int Min(struct Array arr)
139: {
140:     int min=arr.A[0];
141:     int i;
142:     for(i=1;i<arr.length;i++)
143:     {
144:         if(arr.A[i]<min)
145:             min=arr.A[i];
146:     }
147:     return min;
148: }
149:
150: int Sum(struct Array arr)
151: {
152:     int s=0;
153:     int i;
154:     for(i=0;i<arr.length;i++)
155:         s+=arr.A[i];
156:
```

```

157:  return s;
158: }
159:
160: float Avg(struct Array arr)
161: {
162:  return (float)Sum(arr)/arr.length;
163: }
164:
165: void Reverse(struct Array *arr)
166: {
167:  int *B;
168:  int i,j;
169:
170:  B=(int *)malloc(arr->length*sizeof(int));
171:  for(i=arr->length-1,j=0;i>=0;i--,j++)
172:   B[j]=arr->A[i];
173:  for(i=0;i<arr->length;i++)
174:   arr->A[i]=B[i];
175:
176: }
177:
178: void Reverse2(struct Array *arr)
179: {
180:  int i,j;
181:  for(i=0,j=arr->length-1;i<j;i++,j--)
182:  {
183:   swap(&arr->A[i],&arr->A[j]);
184:  }
185: }
186:
187: void InsertSort(struct Array *arr,int x)
188: {
189:  int i=arr->length-1;
190:  if(arr->length==arr->size)
191:   return;
192:  while(i>=0 && arr->A[i]>x)
193:  {
194:   arr->A[i+1]=arr->A[i];
195:   i--;

```

```

196: }
197: arr->A[i+1]=x;
198: arr->length++;
199:
200: }
201:
202: int isSorted(struct Array arr)
203: {
204:     int i;
205:     for(i=0;i<arr.length-1;i++)
206:     {
207:         if(arr.A[i]>arr.A[i+1])
208:             return 0;
209:     }
210:     return 1;
211: }
212:
213: void Rearrange(struct Array *arr)
214: {
215:     int i,j;
216:     i=0;
217:     j=arr->length-1;
218:
219:     while(i<j)
220:     {
221:         while(arr->A[i]<0)i++;
222:         while(arr->A[j]>=0)j--;
223:         if(i<j)swap(&arr->A[i],&arr->A[j]);
224:     }
225:
226: }
227:
228: struct Array* Merge(struct Array *arr1,struct Array *arr2)
229: {
230:     int i,j,k;
231:     i=j=k=0;
232:
233:     struct Array *arr3=(struct Array *)malloc(sizeof(struct Array));
234:

```

```

235: while(i<arr1->length && j<arr2->length)
236: {
237:   if(arr1->A[i]<arr2->A[j])
238:     arr3->A[k++]=arr1->A[i++];
239:   else
240:     arr3->A[k++]=arr2->A[j++];
241: }
242: for(;i<arr1->length;i++)
243:   arr3->A[k++]=arr1->A[i];
244: for(;j<arr2->length;j++)
245:   arr3->A[k++]=arr2->A[j];
246: arr3->length=arr1->length+arr2->length;
247: arr3->size=10;
248:
249: return arr3;
250: }
251:
252: struct Array* Union(struct Array *arr1,struct Array *arr2)
253: {
254:   int i,j,k;
255:   i=j=k=0;
256:
257:   struct Array *arr3=(struct Array *)malloc(sizeof(struct Array));
258:
259:   while(i<arr1->length && j<arr2->length)
260:   {
261:     if(arr1->A[i]<arr2->A[j])
262:       arr3->A[k++]=arr1->A[i++];
263:     else if(arr2->A[j]<arr1->A[i])
264:       arr3->A[k++]=arr2->A[j++];
265:     else
266:     {
267:       arr3->A[k++]=arr1->A[i++];
268:       j++;
269:     }
270:   }
271:   for(;i<arr1->length;i++)
272:     arr3->A[k++]=arr1->A[i];
273:   for(;j<arr2->length;j++)

```

```
274:  arr3->A[k++]=arr2->A[j];
275:
276:  arr3->length=k;
277:  arr3->size=10;
278:
279:  return arr3;
280: }
281:
282: struct Array* Intersection(struct Array *arr1,struct Array *arr2)
283: {
284:     int i,j,k;
285:     i=j=k=0;
286:
287:     struct Array *arr3=(struct Array *)malloc(sizeof(struct Array));
288:
289:     while(i<arr1->length && j<arr2->length)
290:     {
291:         if(arr1->A[i]<arr2->A[j])
292:             i++;
293:         else if(arr2->A[j]<arr1->A[i])
294:             j++;
295:         else if(arr1->A[i]==arr2->A[j])
296:         {
297:             arr3->A[k++]=arr1->A[i++];
298:             j++;
299:         }
300:     }
301:
302:     arr3->length=k;
303:     arr3->size=10;
304:
305:     return arr3;
306: }
307:
308: struct Array* Difference(struct Array *arr1,struct Array *arr2)
309: {
310:     int i,j,k;
311:     i=j=k=0;
312:
```



```

313:  struct Array *arr3=(struct Array*)malloc(sizeof(struct Array));
314:
315:  while(i<arr1->length && j<arr2->length)
316:  {
317:  if(arr1->A[i]<arr2->A[j])
318:  arr3->A[k++]=arr1->A[i++];
319:  else if(arr2->A[j]<arr1->A[i])
320:  j++;
321:  else
322:  {
323:  i++;
324:  j++;
325:  }
326:  }
327:  for(;i<arr1->length;i++)
328:  arr3->A[k++]=arr1->A[i];
329:
330:
331:  arr3->length=k;
332:  arr3->size=10;
333:
334:  return arr3;
335: }
336:
337: int main()
338: {
339:  struct Array arr1;
340:  int ch;
341:  int x,index;
342:
343:  printf("Enter Size of Array:");
344:  scanf("%d",&arr1.size);
345:  arr1.A=(int *)malloc(arr1.size*sizeof(int));
346:  arr1.length=0;
347:  do
348:  {
349:  printf("\n\nMenu\n");
350:  printf("1. Insert\n");
351:  printf("2. Delete\n");

```

```
352: printf("3. Search\n");
353: printf("4. Sum\n");
354: printf("5. Display\n");
355: printf("6.Exit\n");
356:
357: printf("Enter you choice : ");
358: scanf("%d",&ch);
359:
360: switch(ch)
361: {
362: case 1: printf("Enter an element and index:");
363: scanf("%d,%d",&x,&index);
364: Insert(&arr1,index,x);
365: break;
366: case 2: printf("Enter index:");
367: scanf("%d",&index);
368: x=Delete(&arr1,index);
369: printf("Deleted Element is %d\n",x);
370: break;
371: case 3:printf("Enter element to search:");
372: scanf("%d",&x);
373: index=LinearSearch(&arr1,x);
374: printf("Element index %d",index);
375: break;
376: case 4:printf("Sum is %d\n",Sum(arr1));
377: break;
378: case 5:Display(arr1);
379:
380: }
381: }while(ch<6);
382: return 0;
383: }
```