

```

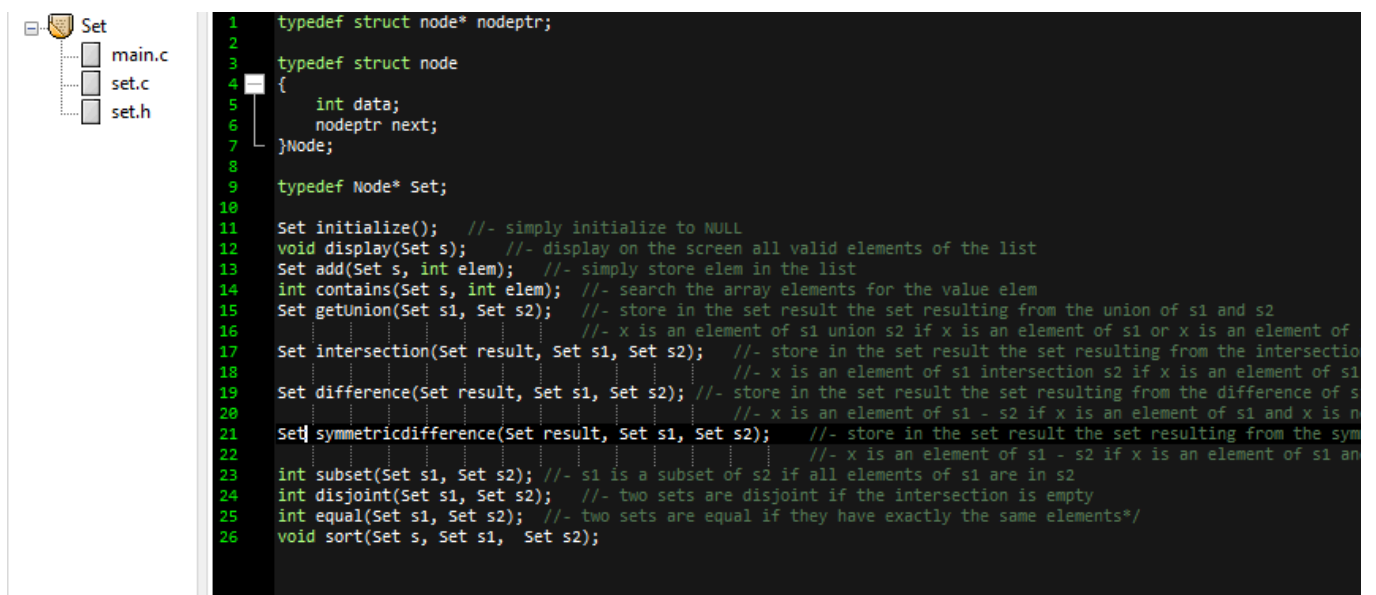
74     s3 = difference(result, s1, s2);
75     printf("Difference: ");
76     display(s3);
77     break;
78     case 5:
79         s3 = symmetricdifference(result, s1, s2);
80         printf("Symmetric Difference: ");
81         display(s3);
82         break;
83     case 6:
84         if(subset(s1, s2) == 1)
85             printf("Set 1 is a subset of set 2");
86         else
87             printf("Set 1 is not a subset of set 2");
88         printf("\n");
89         break;
90     case 7:
91         if(disjoint(s1, s2) == 1)
92             printf("Both sets are disjoint");
93         else
94             printf("Both sets are not disjoint");
95         printf("\n");
96         break;
97     case 8:
98         if(equal(s1, s2) == 1)
99             printf("Both sets are equal");
100        else
101            printf("Both sets are not equal");
102        printf("\n");
103        break;
104    default:
105        printf("Invalid Input");
106        getch(x);
107    }
108    printf("\n");
109    system("pause");
110}
111
112return 0;
113}

```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Kaye\Documents\BSCS\2nd Year\CSIT221\Activities\Structur
- Output Size: 133.3466796875 KiB
- Compilation Time: 0.17s

Figure 1. main.c



```

1  typedef struct node* nodeptr;
2
3  typedef struct node
4  {
5      int data;
6      nodeptr next;
7  }Node;
8
9  typedef Node* Set;
10
11  Set initialize(); // simply initialize to NULL
12  void display(Set s); // display on the screen all valid elements of the list
13  Set add(Set s, int elem); // simply store elem in the list
14  int contains(Set s, int elem); // search the array elements for the value elem
15  Set getUnion(Set s1, Set s2); // store in the set result the set resulting from the union of s1 and s2
16  // x is an element of s1 union s2 if x is an element of s1 or x is an element of s2
17  Set intersection(Set result, Set s1, Set s2); // store in the set result the set resulting from the intersection
18  // x is an element of s1 intersection s2 if x is an element of s1 and x is an element of s2
19  Set difference(Set result, Set s1, Set s2); // store in the set result the set resulting from the difference of s1 and s2
20  // x is an element of s1 - s2 if x is an element of s1 and x is not an element of s2
21  Set symmetricdifference(Set result, Set s1, Set s2); // store in the set result the set resulting from the symmetric difference
22  // x is an element of s1 - s2 if x is an element of s1 and x is not an element of s2, or x is an element of s2 and x is not an element of s1
23  int subset(Set s1, Set s2); // s1 is a subset of s2 if all elements of s1 are in s2
24  int disjoint(Set s1, Set s2); // two sets are disjoint if the intersection is empty
25  int equal(Set s1, Set s2); // two sets are equal if they have exactly the same elements*
26  void sort(Set s, Set s1, Set s2);

```

Figure 2. set.h

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "set.h"
4
5  Set initialize()
6  {
7      Set head = NULL;
8
9      return head;
10 }
11
12 void display(Set s)
13 {
14     nodeptr ptr=s;//initialize
15
16     while(ptr!=NULL)
17     { //condition
18         printf("%d ",ptr->data);//task
19         ptr = ptr->next;//update
20     }
21     printf("\n");
22 }
23
24 int contains(Set s, int elem)
25 {
26     int found=0;
27     nodeptr ptr=s;
28
29
30     while(ptr!=NULL)
31     {
32         if(ptr->data==elem)
33         {
34             found=1;
35             break;
36         }
37         ptr = ptr->next;
38     }
39
40     return found;
41 }
42
43 void swap(Set a, Set b)
44 {
45     int temp = a->data;
46     a->data = b->data;
47     b->data = temp;
48 }
49
50 Set add(Set s, int elem)
51 {
52     if(contains(s,elem)==0)
53     {
54         nodeptr temp = (nodeptr)malloc(sizeof(Node));
55         temp->data = elem;
56         temp->next = s;
57         s = temp;
58     }
59
60     int swapped, i;
61     nodeptr ptr1;
62     nodeptr lptr = NULL;
63
64     //Checks empty list
65     if(s == NULL)
66         return;
67
68     //Sorts into ascending order
69     do
70     {
71         swapped = 0;
72         ptr1 = s;
73
74         while(ptr1->next != lptr)
75         {
76             if(ptr1->data > ptr1->next->data)
77             {
78                 swap(ptr1, ptr1->next);
79                 swapped = 1;
80             }

```

```

81         ptr1 = ptr1->next;
82     }
83     lptr = ptr1;
84 }
85 while(swapped);
86
87 return s;
88 }
89
90
91
92 Set getUnion(Set s1, Set s2)
93 {
94     Set result = initialize();
95
96     //traverse s1 and add to result
97     nodeptr ptr = s1;
98
99     while(ptr!=NULL)
100     {
101         result=add(result,ptr->data);
102         ptr=ptr->next;
103     }
104
105     //traverse s2 and add to result
106     ptr = s2;
107
108     while(ptr!=NULL)
109     {
110         result=add(result,ptr->data);
111         ptr=ptr->next;
112     }
113
114     return result;
115 }
116
117 Set intersection(Set result, Set s1, Set s2)
118 {
119     nodeptr ptemp, qtemp, itemp, irear, ifront;
120
121     result = initialize();
122
123     ptemp = s1;
124     while (ptemp != NULL)
125     {
126         qtemp = s2;
127         ifront = result;
128         while (qtemp != NULL && ptemp->data != qtemp->data)
129         {
130             qtemp = qtemp->next;
131         }
132         if (qtemp != NULL)
133         {
134             if (ifront != NULL)
135             {
136                 if (ifront->data == qtemp->data)
137                 {
138                     ptemp = ptemp->next;
139                     continue;
140                 }
141                 ifront = ifront->next;
142             }
143             itemp = (nodeptr)malloc(sizeof(Node));
144             itemp->data = qtemp->data;
145             itemp->next = NULL;
146             if (result == NULL)
147             {
148                 result = itemp;
149             }
150             else
151             {
152                 irear->next = itemp;
153             }
154             irear = itemp;
155         }
156         ptemp = ptemp->next;
157     }
158
159     return result;
160 }

```

```

163 Set difference(Set result, Set s1, Set s2)
164 {
165     result = initialize();
166
167     nodeptr ptemp, qtemp;
168
169     ptemp = s1;
170     qtemp = s2;
171     while(ptemp != NULL && qtemp != NULL)
172     {
173         if(ptemp->data == qtemp->data)
174         {
175             ptemp = ptemp->next;
176             qtemp = qtemp->next;
177         }
178         else
179         {
180             if(ptemp->data < qtemp->data)
181             {
182                 result = add(result, ptemp->data);
183                 ptemp = ptemp->next;
184             }
185             else
186                 qtemp = qtemp->next;
187         }
188     }
189
190     while(ptemp != NULL)
191     {
192         result = add(result, ptemp->data);
193         ptemp = ptemp->next;
194     }
195
196     return result;
197 }
198
199 Set symmetricdifference(Set result, Set s1, Set s2)
200 {
201     result = initialize();
202
203     //Traverse s1 and s2
204     while(s1 != NULL && s2 != NULL)
205     {
206         //Checks what's not in set 2 and adds it to the result
207         if(s1->data < s2->data)
208         {
209             result = add(result, s1->data);
210             s1 = s1->next;
211         }
212         //Checks what's not in Set 1 and adds it to the result
213         else if(s2->data < s1->data)
214         {
215             result = add(result, s2->data);
216             s2 = s2->next;
217         }
218         else
219         {
220             s1 = s1->next;
221             s2 = s2->next;
222         }
223     }
224
225     return result;
226 }
227
228 int subset(Set s1, Set s2)
229 {
230     nodeptr ptemp = s1;
231     nodeptr qtemp = s2;
232
233     //If both lists are empty, return true
234     if(s1 == NULL && s2 == NULL)
235         return 1;
236
237     //If either of the set is empty, return false
238     if(s1 == NULL || s1 != NULL && s2 == NULL)
239         return 0;
240
241     //Traverse s2
242     while(s2 != NULL)

```

```

243 {
244     //Initialize qtemp with the current node of s2
245     qtemp = s2;
246
247     //Starts matching s1 with s2
248     while(ptemp != NULL)
249     {
250         //Return false if s2 is empty and s1 is not
251         if(qtemp == NULL)
252             return 0;
253
254         //If data is same, proceed
255         else if(ptemp->data == qtemp->data)
256         {
257             ptemp = ptemp->next;
258             qtemp = qtemp->next;
259         }
260
261         //Break loop if no equal found
262         else
263             break;
264     }
265
266     //Returns true if first list is gets traversed
267     if(ptemp == NULL)
268         return 1;
269
270     //Initialize s1 with ptemp again
271     ptemp = s1;
272
273     //Move to the next node of s2
274     s2 = s2->next;
275 }
276
277 return 0;
278 }
279
280 int disjoint(Set s1, Set s2)
281 {
282     Set result;
283
284     nodeptr ptemp, qtemp, itemp, irear, ifront;
285
286     result = initialize();
287
288     ptemp = s1;
289     while (ptemp != NULL)
290     {
291         qtemp = s2;
292         ifront = result;
293         while (qtemp != NULL && ptemp->data != qtemp->data)
294         {
295             qtemp = qtemp->next;
296         }
297         if (qtemp != NULL)
298         {
299             if (ifront != NULL)
300             {
301                 if (ifront->data == qtemp->data)
302                 {
303                     ptemp = ptemp->next;
304                     continue;
305                 }
306                 ifront = ifront->next;
307             }
308             itemp = (nodeptr)malloc(sizeof(Node));
309             itemp->data = qtemp->data;
310             itemp->next = NULL;
311             if (result == NULL)
312             {
313                 return 1;
314             }
315             else
316             {
317                 irear->next = itemp;
318             }
319             irear = itemp;
320         }
321         ptemp = ptemp->next;
322     }
323 }

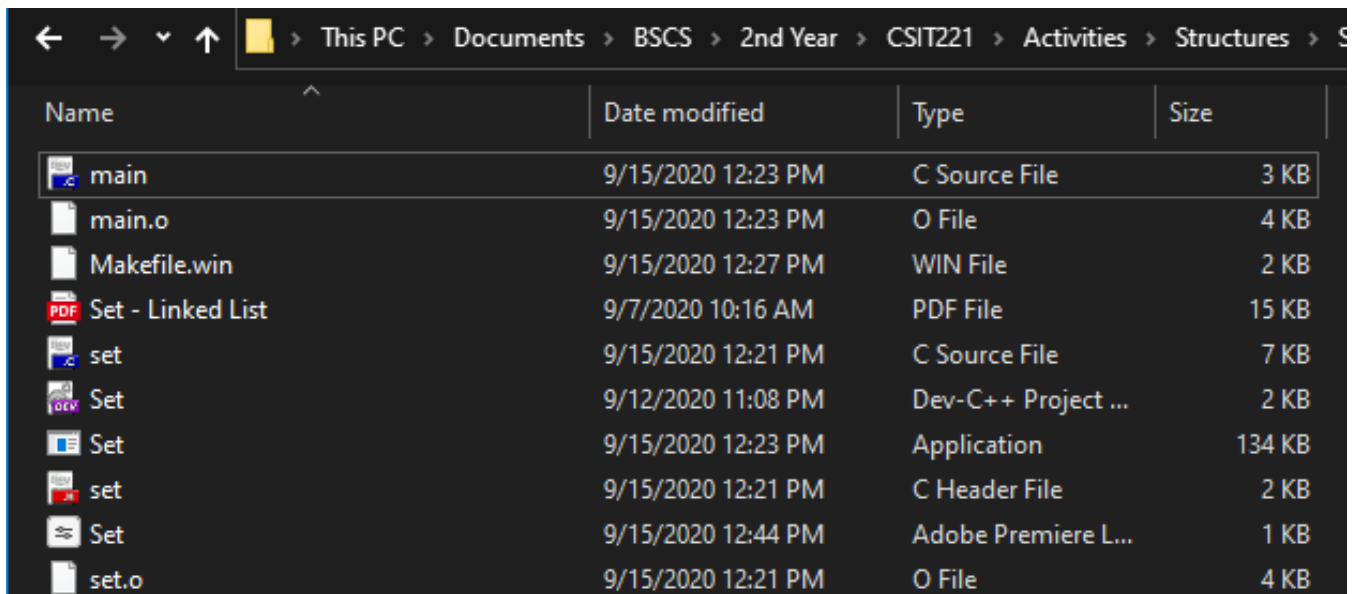
```

```

324     return 0;
325 }
326
327 int equal(Set s1, Set s2)
328 {
329     nodeptr ptemp = s1;
330     nodeptr qtemp = s2;
331
332     //If both lists are empty, return true
333     if(s1 == NULL && s2 == NULL)
334         return 1;
335
336     //If set 1 is empty and s2 is not, return false
337     if(s1 == NULL || s1 != NULL && s2 == NULL)
338         return 0;
339
340     //Traverse s2
341     while(s2 != NULL)
342     {
343         //Initialize qtemp with the current node of s2
344         qtemp = s2;
345
346         //Starts matching s1 with s2
347         while(ptemp != NULL)
348         {
349             //Return false if s2 is empty and s1 is not
350             if(qtemp == NULL)
351                 return 0;
352
353             //Returns true if both sets are equal
354             else if(ptemp->data == qtemp->data)
355             {
356                 return 1;
357             }
358
359             //Break loop if no equal found
360             else
361                 break;
362         }
363
364         //Returns true if first list is gets traversed
365         if(ptemp == NULL)
366             return 1;
367
368         //Initialize s1 with ptemp again
369         ptemp = s1;
370
371         //Move to the next node of s2
372         s2 = s2->next;
373     }
374
375     return 0;
376 }

```

Figure 3. set.c



Name	Date modified	Type	Size
main	9/15/2020 12:23 PM	C Source File	3 KB
main.o	9/15/2020 12:23 PM	O File	4 KB
Makefile.win	9/15/2020 12:27 PM	WIN File	2 KB
Set - Linked List	9/7/2020 10:16 AM	PDF File	15 KB
set	9/15/2020 12:21 PM	C Source File	7 KB
Set	9/12/2020 11:08 PM	Dev-C++ Project ...	2 KB
Set	9/15/2020 12:23 PM	Application	134 KB
set	9/15/2020 12:21 PM	C Header File	2 KB
Set	9/15/2020 12:44 PM	Adobe Premiere L...	1 KB
set.o	9/15/2020 12:21 PM	O File	4 KB

Figure 4. Directory of the Files