Set - [Set.dev] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit

(globals)

Project Class

Set
- main.c
- set.c
- set.h

main.c set.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "set.h"

/* run this program using the console pauser or add your own getch, system("pause") or input loop */

int main(int argc, char *argv[])
{
    Set s1, s2, s3, s, result;
    char x;
    int elem;

    s1 = initialize();
    s2 = initialize();
    s3 = initialize();

    while(1)
    {
        system("cls");

        //Menu
        printf("Set Menu\n\n");
        printf("[1] Add\n");
        printf("[2] Union\n");
        printf("[3] Intersection\n");
        printf("[4] Difference\n");
        printf("[5] Symmetric Difference\n");
        printf("[6] Subset\n");
        printf("[7] Disjoint\n");
        printf("[8] Equal");
        printf("\n\nEnter choice: ");
        int choice; scanf("%d", &choice);

        switch(choice)
        {
            case 1:
                system("cls");
                printf("Add Menu\n\n");
                printf("[a] Set 1\n");
                printf("[b] Set 2");
```

Set
- main.c
- set.c
- set.h

```c
                printf("\n\nEnter choice: ");
                char choicey; scanf("%s", &choicey);

                if(choicey == 'a')
                {
                    printf("Enter element: ");
                    scanf("%d", &elem);
                    s1 = add(s1, elem);
                    printf("Set 1: ");
                    display(s1);
                }
                else if(choicey == 'b')
                {
                    printf("Enter element: ");
                    scanf("%d", &elem);
                    s2 = add(s2, elem);
                    printf("Set 2: ");
                    display(s2);
                }
                else
                    printf("Invalid choice");
                break;
            case 2:
                s3 = getUnion(s1, s2);
                printf("Union: ");
                display(s3);
                break;
            case 3:
                s3 = intersection(result, s1, s2);
                printf("Intersection: ");
                display(s3);
                break;
            case 4:
                s3 = difference(result, s1, s2);
                printf("Difference: ");
                display(s3);
                break;
            case 5:
                s3 = symmetricdifference(result, s1, s2);
                printf("Symmetric Difference: ");
```

Set
- main.c
- set.c
- set.h

```c
                s3 = difference(result, s1, s2);
                printf("Difference: ");
                display(s3);
                break;
            case 5:
                s3 = symmetricdifference(result, s1, s2);
                printf("Symmetric Difference: ");
                display(s3);
                break;
            case 6:
                if(subset(s1, s2) == 1)
                    printf("Set 1 is a subset of set 2");
                else
                    printf("Set 1 is not a subset of set 2");
                printf("\n");
                break;
            case 7:
                if(disjoint(s1, s2) == 1)
                    printf("Both sets are disjoint");
                else
                    printf("Both sets are not disjoint");
                printf("\n");
                break;
            case 8:
                if(equal(s1, s2) == 1)
                    printf("Both sets are equal");
                else
                    printf("Both sets are not equal");
                printf("\n");
                break;
            default:
                printf("Invalid Input");
                getch(x);
        }
        printf("\n");
        system("pause");
    }

    return 0;
}
```

Compiler | Resources | Compile Log | Debug | Find Results | Close

Abort Compilation

Shorten compiler paths

```
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

Set
- main.c
- set.c
- set.h

```c
typedef struct node* nodeptr;

typedef struct node
{
    int data;
    nodeptr next;
}Node;

typedef Node* Set;

Set initialize();   //- simply initialize to NULL
void display(Set s);    //- display on the screen all valid elements of the list
Set add(Set s, int elem);   //- simply store elem in the list
int contains(Set s, int elem);  //- search the array elements for the value elem
Set getUnion(Set s1, Set s2);   //- store in the set result the set resulting from the union of s1 and s2
                                //- x is an element of s1 union s2 if x is an element of s1 or x is an element of
Set intersection(Set result, Set s1, Set s2);   //- store in the set result the set resulting from the intersectio
                                                //- x is an element of s1 intersection s2 if x is an element of s1
Set difference(Set result, Set s1, Set s2); //- store in the set result the set resulting from the difference of s
                                            //- x is an element of s1 - s2 if x is an element of s1 and x is n
Set symmetricdifference(Set result, Set s1, Set s2);    //- store in the set result the set resulting from the sym
                                                        //- x is an element of s1 - s2 if x is an element of s1 an
int subset(Set s1, Set s2); //- s1 is a subset of s2 if all elements of s1 are in s2
int disjoint(Set s1, Set s2);   //- two sets are disjoint if the intersection is empty
int equal(Set s1, Set s2);  //- two sets are equal if they have exactly the same elements*/
void sort(Set s, Set s1,  Set s2);
```

Figure 2. set.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "set.h"

Set initialize()
{
    Set head = NULL;

    return head;
}

void display(Set s)
{
    nodeptr ptr=s;//initialize

    while(ptr!=NULL)
    {//condition
        printf("%d ",ptr->data);//task
        ptr = ptr->next;//update
    }
    printf("\n");
}

int contains(Set s, int elem)
{
    int found=0;
    nodeptr ptr=s;


    while(ptr!=NULL)
    {
        if(ptr->data==elem)
        {
            found=1;
            break;
        }
        ptr = ptr->next;
    }

    return found;
}

void swap(Set a, Set b)
{
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}

Set add(Set s, int elem)
{
    if(contains(s,elem)==0)
    {
        nodeptr temp = (nodeptr)malloc(sizeof(Node));
        temp->data = elem;
        temp->next = s;
        s = temp;
    }

    int swapped, i;
    nodeptr ptr1;
    nodeptr lptr = NULL;

    //Checks empty list
    if(s == NULL)
        return;

    //Sorts into ascending order
    do
    {
        swapped = 0;
        ptr1 = s;

        while(ptr1->next != lptr)
        {
            if(ptr1->data > ptr1->next->data)
            {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
```

```
            ptr1 = ptr1->next;
        }
        lptr =ptr1;
    }
    while(swapped);

    return s;
}



Set getUnion(Set s1, Set s2)
{
    Set result = initialize();

    //traverse s1 and add to result
    nodeptr ptr = s1;

    while(ptr!=NULL)
    {
        result=add(result,ptr->data);
        ptr=ptr->next;
    }

    //traverse s2 and add to result
    ptr = s2;

    while(ptr!=NULL)
    {
        result=add(result,ptr->data);
        ptr=ptr->next;
    }

    return result;
}

Set intersection(Set result, Set s1, Set s2)
{
    nodeptr ptemp, qtemp, itemp, irear, ifront;

    result = initialize();

    ptemp = s1;
    while (ptemp != NULL)
    {
        qtemp = s2;
        ifront = result;
        while (qtemp != NULL && ptemp->data != qtemp->data)
        {
            qtemp = qtemp->next;
        }
        if (qtemp != NULL)
        {
            if (ifront != NULL)
            {
                if (ifront->data == qtemp->data)
                {
                    ptemp = ptemp->next;
                    continue;
                }
                ifront = ifront->next;
            }
            itemp = (nodeptr)malloc(sizeof(Node));
            itemp->data = qtemp->data;
            itemp->next = NULL;
            if (result == NULL)
            {
                result = itemp;
            }
            else
            {
                irear->next = itemp;
            }
            irear = itemp;
        }
        ptemp = ptemp->next;
    }

    return result;
}
```

```
Set difference(Set result, Set s1, Set s2)
{
    result = initialize();

    nodeptr ptemp, qtemp;

    ptemp = s1;
    qtemp = s2;
    while(ptemp != NULL && qtemp != NULL)
    {
        if(ptemp->data == qtemp->data)
        {
            ptemp = ptemp->next;
            qtemp = qtemp->next;
        }
        else
        {
            if(ptemp->data < qtemp->data)
            {
                result = add(result, ptemp->data);
                ptemp = ptemp->next;
            }
            else
                qtemp = qtemp->next;
        }
    }

    while(ptemp != NULL)
    {
        result = add(result, ptemp->data);
        ptemp = ptemp->next;
    }

    return result;
}

Set symmetricdifference(Set result, Set s1, Set s2)
{
    result = initialize();
```

```
    //Traverse s1 and s2
    while(s1 != NULL && s2 != NULL)
    {
        //Checks what's not in set 2 and adds it to the result
        if(s1->data < s2->data)
        {
            result = add(result, s1->data);
            s1 = s1->next;
        }
        //Checks whats not in Set 1 and adds it to the result
        else if(s2->data < s1->data)
        {
            result = add(result, s2->data);
            s2 = s2->next;
        }
        else
        {
            s1 = s1->next;
            s2 = s2->next;
        }
    }

    return result;
}

int subset(Set s1, Set s2)
{
    nodeptr ptemp = s1;
    nodeptr qtemp = s2;

    //If both lists are empty, return true
    if(s1 == NULL && s2 == NULL)
        return 1;

    //If either of the set is empty, return false
    if(s1 == NULL || s1 != NULL && s2 == NULL)
        return 0;

    //Traverse s2
    while(s2 != NULL)
```

```
    {
        //Initialize qtemp with the current node of s2
        qtemp = s2;

        //Starts matching s1 with s2
        while(ptemp != NULL)
        {
            //Return false if s2 is empty and s1 is not
            if(qtemp == NULL)
                return 0;

            //If data is same, proceed
            else if(ptemp->data == qtemp->data)
            {
                ptemp = ptemp->next;
                qtemp = qtemp->next;
            }

            //Break loop if no equal found
            else
                break;
        }

        //Returns true if first list is gets traversed
        if(ptemp == NULL)
            return 1;

        //Initialize s1 with ptemp again
        ptemp = s1;

        //Move to the next node of s2
        s2 = s2->next;
    }

    return 0;
}

int disjoint(Set s1, Set s2)
{
    Set result;
```

```
    nodeptr ptemp, qtemp, itemp, irear, ifront;

    result = initialize();

    ptemp = s1;
    while (ptemp != NULL)
    {
        qtemp = s2;
        ifront = result;
        while (qtemp != NULL && ptemp->data != qtemp->data)
        {
            qtemp = qtemp->next;
        }
        if (qtemp != NULL)
        {
            if (ifront != NULL)
            {
                if (ifront->data == qtemp->data)
                {
                    ptemp = ptemp->next;
                    continue;
                }
                ifront = ifront->next;
            }
            itemp = (nodeptr)malloc(sizeof(Node));
            itemp->data = qtemp->data;
            itemp->next = NULL;
            if (result == NULL)
            {
                return 1;
            }
            else
            {
                irear->next = itemp;
            }
            irear = itemp;
        }
        ptemp = ptemp->next;
    }
```

```c
    return 0;
}

int equal(Set s1, Set s2)
{
    nodeptr ptemp = s1;
    nodeptr qtemp = s2;

    //If both lists are empty, return true
    if(s1 == NULL && s2 == NULL)
        return 1;

    //If  set 1 is empty and s2 is not, return false
    if(s1 == NULL || s1 != NULL && s2 == NULL)
        return 0;

    //Traverse s2
    while(s2 != NULL)
    {
        //Initialize qtemp with the current node of s2
        qtemp = s2;

        //Starts matching s1 with s2
        while(ptemp != NULL)
        {
            //Return false if s2 is empty and s1 is not
            if(qtemp == NULL)
                return 0;

            //Returns true if both sets are equal
            else if(ptemp->data == qtemp->data)
            {
                return 1;
            }

            //Break loop if no equal found
            else
                break;
        }

        //Returns true if first list is gets traversed
        if(ptemp == NULL)
            return 1;

        //Initialize s1 with ptemp again
        ptemp = s1;

        //Move to the next node of s2
        s2 = s2->next;
    }

    return 0;
}
```

Figure 3. set.c

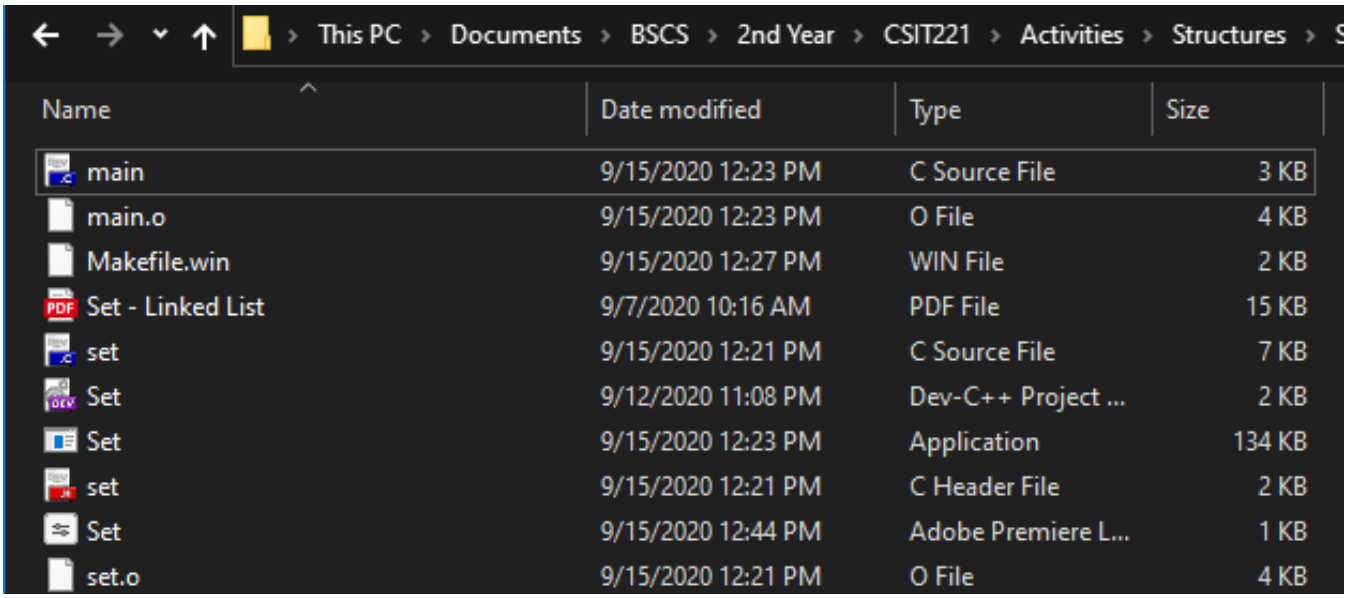

Figure 4. Directory of the Files