תרגיל בית 1 מתם:

חלק יבש:

2.1.1 סעיף א:

שגיאות קונבנציה:

1-שם הפונקציה צריך להיותD) stringDuplicator גדולה , בקוד הנתון האות הראשונה של המילה השנייה היא אות קטנה).

2- שם הארגומנט הראשון s חסר משמעות, צריך לכתוב str

3- שם משתנה LEN צריך להיות כתוב באותיות קטנות len (אותיות גדולות שמורות למאקרו).

4- הבלוק של הלולאת for צריך להיות בהזחה ימינה.

5- השם של הפונקציה צריך להיות פועל לדוגמא: duplicateString

שגיאות תכנות:

1-בשורה השמינית הארגומנט של הפונקציה strlen צריך להיות (s) ולא ( (*s- הפונקציה צריכה לקבל פוינטר.

2- בשורה 9 ההקצאה צריכה להיות בגודל ((len*times)+1) כדי לשמור מקום ל'\0'.

3- בשורה 6 הארגומנט של assert צריך להיות רק s ולא !s. כי רוצים לוודא שs לא null ובצורה הזו התכנית תקרוס אם s לא Null.

4- בשורה 6 ו-10 לא צריך להיות assert .

בשורה 10- כיוון שבדרישות הפונקציה מצוין שצריך להחזיר Null במקרה של שגיאה בריצת הפונקציה. אם המאלוק נכשל בהקצאת הזיכרון זו שגיאה בריצת הפונקציה ולכן במקרה כזה צריך להחזיר Null ולא להקריס את התכנית כמו שassert עושה.

בשורה 6 שוב אם קיבלנו ארגומנט null צריך להחזיר איזו שגיאה שקיבלנו ארגומנט לא תקין ולא להקריס את התכנית. (למשל שניתן ל strlen null זה יגרום לשגיאה).

5-בלולאת for כל איטרציה משנה את הפוינטר out .

בסוף הלולאה out לא מצביע על תחילת המחרוזת שיצרנו וברגע שמחזירים אותו למשתמש לא מקבלים את התוצאה הרצויה.

2.1.2

2.1.2 סעיף ב:

```
char *duplicateString(char *str, int times) {
    if(!str)
    {
        return NULL;
    }
    assert(times > 0);
    int len = strlen(str);
    char *result = malloc((sizeof(char) * len * times)+1);
    if(!result)
        {
            return NULL;
        }
    for (int i = 0; i < times; i++) {
    strcpy(result+(i*len), str);

    }
    result[times*len]='\0';
    return result;
    }
```

2.2:

```
ErrorCode mergeSortedLists(Node list1, Node list2, Node *mergedOut)

{

    if(list1 == NULL || list2 == NULL)

    {

        *mergedOut = NULL;

        return NULL_ARGUMENT;

    }



    if(!isListSorted(list1) || !isListSorted(list2) )

    {

        *mergedOut = NULL;

        return   UNSORTED_LIST;

    }
```

```
Node ptr1 = list1;
Node ptr2 = list2;
Node ptrMerged = NULL;

if((ptr1->x) <= (ptr2->x))
{
    ptrMerged=createNode(ptr1->x);

    if (ptrMerged == NULL)
    {
        return MEMORY_ERROR;
    }

    ptr1=ptr1->next;
}
else
{
    ptrMerged=createNode(ptr2->x);

    if (ptrMerged == NULL)
    {
        return MEMORY_ERROR;
    }

    ptr2=ptr2->next;
}
```

```
    *mergedOut = ptrMerged;




    while (ptr1 != NULL && ptr2 != NULL )
    {
        if((ptr1->x) <= (ptr2->x))
        {
            insertData(&ptr1, &ptrMerged);

            if (ptrMerged == NULL)
            {
                destroyNode(*mergedOut);
                return MEMORY_ERROR;
            }

        }

        else
        {
            insertData(&ptr2, &ptrMerged);

            if (ptrMerged == NULL)
            {
                destroyNode(*mergedOut);
                return MEMORY_ERROR;
            }
```

```
        }
    }

    while (ptr1 != NULL)
    {
        insertData(&ptr1, &ptrMerged);

        if (ptrMerged == NULL)
        {
            destroyNode(*mergedOut);
            return MEMORY_ERROR;
        }

    }

    while (ptr2 != NULL)
    {
        insertData(&ptr2, &ptrMerged);

        if (ptrMerged == NULL)
        {
            destroyNode(*mergedOut);
            return MEMORY_ERROR;
        }
    }

    return SUCCESS;
```

```
}
```