**Parcing**

**פירסוס:** פעולה שלוקחת ערך טקסטואלי והופכת אותו למבנה נתונים פנימי של התוכנית**.**

**פונקציה חדשה:** (str, "<token>")strtok. אסימוןToken=

**איך יתבצע הפירסוס:** לעשות מצביע פוינטר מסוג תו ששווה לפונקצייה(הטוקן צריך להיות התו שעל פיו תהיה החלוקה) מה שנקבל זה החלוקה הראשונה, כדי להמשיך ולקבל את שאר החלוקות יש להציב null במקום str בפונקצייה.

**דוגמא:**

int returnSum(char str[100])

{

    int sum = 0;

    char\* token = strtok(str, ",");

    while (token != NULL)

    {

        sum = sum + atoi(token);

        token = strtok(NULL, ",");

    }

    return sum;

}

**Parcing and structures**

* בשמירה וטעינת קובץ בעזרת פרסינג למבנה יש להשתמש בrb,wb. בפונקציות.

**דוגמא:**

struct OskarActor\_Header

{

    int Version;

    int ActorCount;

    char Reserved[1000];

};

struct OskarActor

{

    int Index;

    int Year;

    char Age;

    int Popularity;

    char Movie[1000];

    char Name[1000];

    struct OskarActor\* next;

    struct OskarActor\* prev;

};

struct OskarActor\* head = NULL;

struct OskarActor\* tail = NULL;

void printList()

{

    struct OskarActor\* currentOskarActor = head;

    printf("Head");

    while (currentOskarActor != NULL)

    {

        printf("->(Index: %d, Year: %d, Age: %d, Popularity: %d, Movie:%s, Name:%s)", currentOskarActor->Index, currentOskarActor->Year, currentOskarActor->Age, currentOskarActor->Popularity, currentOskarActor->Movie, currentOskarActor->Name);

        currentOskarActor = currentOskarActor->next;

    }

    printf("->Null\n");

}

void freeList()

{

    struct OskarActor\* currentOskarActor = head;

    struct OskarActor\* releaseOskarActor = head;

    while (currentOskarActor != NULL)

    {

        releaseOskarActor = currentOskarActor;

        currentOskarActor = currentOskarActor->next;

        free(releaseOskarActor);

    }

}

void resetList()

{

    if (head != NULL)

    {

        freeList();

        head = NULL;

        tail = NULL;

    }

}

Add(int index, int year, int age, int popularity, char movie[1000], char name[1000])

{

    struct OskarActor\* curr;

    curr = (struct OskarActor\*)malloc(sizeof(struct OskarActor));

    curr->Index = index;

    curr->Year = year;

    curr->Age = age;

    curr->Popularity = popularity;

    strcpy(curr->Movie, movie);

    strcpy(curr->Name, name);

    curr->next = NULL;

    curr->prev = NULL;

    if (head == NULL)

    {

        head = curr;

        tail = curr;

    }

    else

    {

        tail->next = curr;

        curr->prev = tail;

        tail = curr;

    }

}

void GetMovieByIndex(int index)

{

    struct OskarActor\* show = NULL;

    struct OskarActor\* curr = head;

    while (curr != NULL)

    {

        if (curr->Index == index)

        {

            show = curr;

            printf("(Index: %d, Year: %d, Age: %d, Popularity: %d, Movie: %s, Name: %s)", show->Index, show->Year, show->Age, show->Popularity, show->Movie, show->Name);

            return;

        }

        curr = curr->next;

    }

    (show) ? printf("(Index: %d, Year: %d, Age: %d, Popularity: %d, Movie: %s, Name: %s)", show->Index, show->Year, show->Age, show->Popularity, show->Movie, show->Name) : printf("this index does not exist in oskar list\n");

}

void PrintAllMoviesForYear(int year)

{

    struct OskarActor\* show = NULL;

    struct OskarActor\* curr = head;

    while (curr != NULL)

    {

        show = curr;

        if (curr->Year == year)

        {

            printf("(Index: %d, Year: %d, Age: %d, Popularity: %d, Movie: %s, Name: %s)", show->Index, show->Year, show->Age, show->Popularity, show->Movie, show->Name);

        }

        curr = curr->next;

    }

    (show && show->Year == year) ? printf("(Index: %d, Year: %d, Age: %d, Popularity: %d, Movie: %s, Name: %s)", show->Index, show->Year, show->Age, show->Popularity, show->Movie, show->Name) : printf("this year does not exist in oskar list\n");

}

void LoadFile()

{

    char movie[1000], name[1000], str[1000];

    int index, age, year;

    FILE\* f = fopen("f.txt", "rb");

    if (f)

    {

        fgets(str, sizeof(str), f);

        while (str[0] != NULL)

        {

            char\* token = strtok(str, ",");

            index = atoi(token);

            while (token != NULL)

            {

                token = strtok(NULL, ",");

                year = atoi(token);

                token = strtok(NULL, ",");

                age = atoi(token);

                token = strtok(NULL, ",");

                strcpy(name, token);

                token = strtok(NULL, ",");

                strcpy(movie, token);

                token = strtok(NULL, ",");

            }

            Add(index, year, age, 0, movie, name);

            str[0] = NULL;

            fgets(str, sizeof(str), f);

        }

        fclose(f);

    }

    else

    {

        // error

        return;

    }

}

int GetOskarActorCount()

{

    int cnt = 0;

    struct OskarActor\* currentOskarActor = head;

    while (currentOskarActor != NULL)

    {

        cnt++;

        currentOskarActor = currentOskarActor->next;

    }

    //printf("\n%d\n",cnt);

    return cnt;

}

void UpdatePopolarity(int inedx)

{

    struct OskarActor\* ret = NULL;

    struct OskarActor\* curr = head;

    while (curr != NULL)

    {

        if (curr->Index == inedx)

        {

            curr->Popularity++;

            ret = curr;

            return 0;

        }

        curr = curr->next;

    }

    printf("index not found in list\n");

}

void run()

{

    int i;

    srand(time(NULL));

    for (int j = 0; j < 100000; j++)

    {

        i = rand();

        printf("%d\n", i);

        UpdatePopolarity(i);

    }

}

void mostPopular()

{

    struct OskarActor\* ret = NULL;

    struct OskarActor\* curr = head;

    int maxPopularity = 0;

    while (curr != NULL)

    {

        if (curr->Popularity > maxPopularity)

        {

            maxPopularity = curr->Popularity;

            ret = curr;

        }

        curr = curr->next;

    }

    printf("\n(Index: %d, Year: %d, Age: %d, Popularity: %d, Movie: %s, Name: %s)", ret->Index, ret->Year, ret->Age, ret->Popularity, ret->Movie, ret->Name);

}

void Save(char filename[100])

{

    struct OskarActor\* curr = head;

    FILE\* f = fopen(filename, "wb");

    struct OskarActor\_Header h;

    h.Version = 1;

    h.ActorCount = GetOskarActorCount();

    if (!f)

    {

        //error

        return 1;

    }

    fwrite(&h, sizeof(struct OskarActor\_Header), 1, f);

    for (int i = 0; i < h.ActorCount; i++)

    {

        fwrite(curr, sizeof(struct OskarActor), 1, f);

        curr = curr->next;

    }

    fclose(f);

}

void Load(char filename[100])

{

    struct OskarActor\_Header headerOfFile;

    //read from file

    FILE\* f = fopen(filename, "rb");

    if (!f)

    {

        //error

        return 1;

    }

    int read = fread(&headerOfFile, sizeof(struct OskarActor\_Header), 1, f);

    if (read == 0)

    {

        //error

        return 1;

    }

    //build the list

    head = NULL;

    tail = NULL;

    struct OskarActor\* curr = (struct OskarActor\*)malloc(sizeof(struct OskarActor));

    for (int i = 0; i < headerOfFile.ActorCount; i++)

    {

        read = fread(curr, sizeof(struct OskarActor), 1, f);

        Add(curr->Index, curr->Year, curr->Age, curr->Popularity, curr->Movie, curr->Name);

    }

    fclose(f);

}