

# Абстрактні типи даних



*Лекція №8*

*Дисципліна «Програмування»*

*2-й семестр*



# Спосіб визначення

Обчислювальні науки пропонують дуже ефективний спосіб визначення нових типів даних, який є процесом переходу від абстрактного до конкретного та складається з трьох етапів:

- 1) Формування абстрактного опису властивостей типу та операцій, які можна виконувати над цим типом. Опис не повинен бути прив'язаним до жодної конкретної реалізації та конкретної мови програмування.

Формальний абстрактний опис подібного роду називають **абстрактним типом даних** (*abstract data type* – ADT).

- 2) Розробка програмного інтерфейсу, який реалізує цей абстрактний тип даних.
- 3) Написання коду для реалізації інтерфейсу.



# Отримання абстракції

Назвемо списком абстрактний тип, що складається з послідовності упорядкованих елементів, кожен з яких містить назву та рейтинг фільму.

Необхідно мати можливість додавати нові елементи в кінець списку та відображати вміст списку.

Корисні операції зі списком для даного проекту:

- ініціалізація списку пустим вмістом;
- додавання елемента в кінець списку;
- з'ясування, чи є список пустим;
- з'ясування, чи є список повним;
- визначення кількості елементів у списку;
- перегляд кожного елемента у списку з метою виконання певної дії, наприклад, відображення елемента списку.



# Отримання абстракції

Більш універсальний перелік операцій зі списками:

- додавання елемента в будь-яке місце списку;
- видалення елемента зі списку;
- отримання елемента зі списку (список залишається незмінним);
- заміна одного елемента у списку іншим;
- пошук елемента у списку.

Абстрактне визначення списку:

**Список** – це об'єкт даних, який може зберігати послідовність елементів, до якого можна застосовувати будь-які з перерахованих раніше операцій.



# Абстрактний тип даних

Прийmemo в якості абстрактного типу даних спрощений список, що містить тільки ті функціональні можливості, які потрібні для проекту «Інформація про фільми».

Короткий опис типу:

**Ім'я типу:** Простий список.

**Властивості типу:** Може містити послідовність елементів.

**Операції типу:**

- 1) Ініціалізація списку пустим вмістом.
- 2) З'ясування, чи є список пустим.
- 3) З'ясування, чи є список повним.
- 4) Визначення кількості елементів у списку.
- 5) Додавання елемента в кінець списку.
- 6) Обхід списку з обробкою кожного елемента.
- 7) Спусташення списку.



# Розробка інтерфейсу

Інтерфейс для простого списку складається зі способу представлення даних та функцій, що реалізують операції абстрактного типу даних.

Проектне рішення інтерфейсу повинне якомога ближче відображати опис АДТ. Воно повинно бути виражено в термінах деякого загального типу **Item**.

Один зі способів досягнення цього передбачає використання засобу **typedef** мови C для визначення **Item** в якості потрібного типу:

```
#define TSIZE 45    // розмір масиву для зберігання назви
struct film
{
    char title[TSIZE];
    int rating;
};
typedef struct film Item;
```



# Розробка інтерфейсу

Якщо пізніше буде потрібен список елементів будь-якої іншої форми даних, можна буде перевизначити тип **Item** і залишити іншу частину визначення інтерфейсу без змін.

Далі необхідно прийняти рішення про спосіб зберігання елементів цього типу.

Застосуємо підхід з використання зв'язаних структур, де кожен зв'язок називається вузлом, що містить необхідну інформацію, та вказівник на наступний вузол.

Щоб підкреслити обрану термінологію, назвемо структуру вузла іменем **node** і застосуємо **typedef**, щоб зробити **Node** іменем типу для структури **node**.

```
typedef struct node
{
    Item item;
    struct node *next;
} Node;
typedef Node *List;
```



# Розробка інтерфейсу

Для керування зв'язним списком потрібен вказівник на його початок, тому ми використовували **typedef**, щоб перетворити **List** на ім'я для вказівника цього типу.

Таким чином, оголошення **List** `movies;` розглядає **movies** як вказівник на зв'язний список.

Для відстеження кількості записів можна використовувати альтернативне визначення списку:

```
typedef struct list
{
    Node *head;      // вказівник на заголовок списку
    int size;         // кількість записів у списку
} List;              // альтернативне визначення списку
```

Тепер оголошення **List** `movies;` слід розглядати як визначення списку, а не встановлення вказівника на вузол або структуру.





# Розробка інтерфейсу

Точне представлення даних списку **movies** є деталлю реалізації, яка не повинна бути помітною на рівні інтерфейсу. Наприклад, під час запуску програма повинна ініціалізувати вказівник на заголовок значенням **NULL**, але не слід застосовувати код на зразок: `movies = NULL;`

Це пов'язано з тим, що у подальшому може з'ясуватися, що реалізація типу **List** у вигляді структури підходить більше, і тоді буде потрібна наступна ініціалізація:

```
movies.next = NULL; movies.size = 0;
```

Замість цього повинна бути можливість записати, наприклад: `InitializeList(movies);`

Програмістам треба знати, що для ініціалізації списку вони повинні застосовувати функцію `InitializeList()`. Вони не зобов'язані знати точну реалізацію даних для змінної **List**.

Це є прикладом **приховування даних** – мистецтва маскування подробиць представлення даних від більш високих рівнів програмування.



# Розробка інтерфейсу

Для надання рекомендацій користувачу прототип функції можна супроводжувати наступними рядками:

```
//=====
// Операція:      ініціалізація списку
// Передумова:    plist вказує на список list
// Постумова:     список ініціалізований пустим змістом
//=====
void InitializeList(List *plist);
```

- 1) Коментарі описують передумови, тобто умови, які повинні бути задовільнені до виклику функції.
- 2) Коментарі описують постумови – умови, які повинні бути задовільнені після виконання функції.
- 3) В якості свого аргументу функція використовує вказівник на список, а не сам список, тому виклик функції буде мати такий вигляд:

```
InitializeList(&movies);
```



# Розробка інтерфейсу

Прийнятий в мові c метод об'єднання інформації про тип і функції до єдиного пакету передбачає розміщення визначень для типу та прототипів функцій (в тому числі коментарів з перед- і постумовами) у файлі заголовку. Цей файл повинен надавати всю інформацію, яка необхідна програмісту для використання типу. Файл заголовку `list.h` для простого типу **List** має наступний вигляд:

```
//=====
// list.h - файл заголовку для простого типу списку
//=====
#ifndef LIST_H_
#define LIST_H_
#include <stdbool.h>          // Функціональна можливість C99

//=====
// Оголошення, які є специфічними для програми
//=====
```



# Розробка інтерфейсу

```
#define TSIZE 45    // розмір масиву для зберігання назви
struct film
{
    char title[TSIZE];
    int rating;
};

//=====
// Оголошення спільних типів
//=====
typedef struct film Item;
typedef struct node
{
    Item item;
    struct node *next;
} Node;
typedef Node *List;
```



# Розробка інтерфейсу

```
//=====
// Прототипи функцій
//=====
// Операція:    ініціалізація списку
// Передумова:  plist вказує на список list
// Постумова:   список ініціалізований пустим змістом
//=====
void InitializeList(List *plist);

//=====
// Операція:    визначення, чи є список пустим
// Передумова:  plist вказує на ініціалізований список
// Постумови:   функція повертає значення True, якщо список
//              пустий, і False в іншому випадку
//=====
bool ListIsEmpty(const List *plist);
```



# Розробка інтерфейсу

```
//=====
// Операція:    визначення, чи є список повним
// Передумова:  plist вказує на ініціалізований список
// Постумови:   функція повертає значення True, якщо список
//              повний, і False в іншому випадку
//=====
bool ListIsFull(const List *plist);

//=====
// Операція:    визначення кількості елементів у списку
// Передумова:  plist вказує на ініціалізований список
// Постумови:   функція повертає кількість елементів у списку
//=====
unsigned int ListItemCount(const List *plist);
```



# Розробка інтерфейсу

```
//=====
// Операція:   додавання елемента в кінець списку
// Передумови: item — елемент, що додається до списку
//              plist вказує на ініціалізований список
// Постумови:  якщо це можливо, функція додає елемент в
//              кінець списку і повертає значення True;
//              в іншому випадку повертає значення False
//=====
bool AddItem(Item item, List *plist);
//=====
// Операція:   застосування функції до кожного елемента списку
// Передумови: plist вказує на ініціалізований список
//              rfun вказує на функцію, яка приймає аргумент
//              Item і не має значення, що повертається
// Постумови:  функція, на яку вказує rfun, виконується один
//              раз для кожного елемента в списку
//=====
void Traverse(const List *plist, void (* pfun)(Item item));
```



# Розробка інтерфейсу

```
//=====
// Операція:      звільнення виділеної пам'яті, якщо вона є
// Передумова:    plist вказує на ініціалізований список
// Постумови:     будь-яка пам'ять, що виділяється для списку,
//               звільняється, і список встановлюється
//               в пустий стан
//=====
void EmptyTheList(List *plist);
#endif
```

В файлі імена функцій починаються з прописних літер для їх позначення як частини інтерфейсного пакету. Крім того, для захисту від множинного включення файлу застосовується прийом з `#ifndef`. Якщо ваш компілятор не підтримує тип `bool` зі стандарту `C99`, можете замінити у файлі заголовку рядок

```
#include <stdbool.h>           // функціональна можливість C99
На enum bool { false, true }; // визначення bool як типу,
                               // і false, true - як значень
```





# Використання інтерфейсу

Оскільки інтерфейс визначений в термінах типів **List** і **Item**, програма повинна бути створена із застосуванням саме цих типів.

Один з можливих планів написання програми:

- 1) Створити змінну **List**.
- 2) Створити змінну **Item**.
- 3) Ініціалізувати список пустим змістом.
- 4) Поки список не заповнений та є вхідні дані:
  - прочитати вхідні дані та помістити їх до змінної **Item**;
  - додати елемент в кінець списку.
- 5) Переглянути кожен елемент списку та відобразити його.



# Реалізація програми

```
//=====
// main.c - файл для використання зв'язного списку в стилі ADT
//=====
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>          // прототип для exit()
#include "list.h"           // визначення List, Item

void showmovies(Item item);
char *s_gets(char *st, int n);

int main(void)
{
    List movies;
    Item temp;

    SetConsoleOutputCP(1251);
    InitializeList(&movies);
    if(ListIsFull(&movies))
    {
```



# Реалізація програми

```
fprintf(stderr, "Доступна пам'ять відсутня! "  
        "Програма завершена.\n");  
exit(1);  
}  
  
// Збір та зберігання інформації  
puts("Введіть назву першого фільму:");  
while(s_gets(temp.title, TSIZE) != NULL &&  
        temp.title[0] != '\0')  
{  
    puts("Введіть своє значення рейтингу <0-10>:");  
    scanf("%d", &temp.rating);  
    while(getchar() != '\n')  
        continue;  
    if(AddItem(temp, &movies) == false)  
    {  
        fprintf(stderr, "Проблема з виділенням пам'яті\n");  
        break;  
    }  
}
```



# Реалізація програми

```
if(ListIsFull(&movies))
{
    puts("Список повний.");
    break;
}
puts("Введіть назву наступного фільму");
puts("(або порожній рядок для припинення вводу):");
}
// відображення списку
if(ListIsEmpty(&movies))
    printf("Дані не введені.");
else
{
    printf("=====\n");
    printf("Список фільмів:\n");
    printf("=====\n");
    Traverse(&movies, showmovies);
}
```



# Реалізація програми

```
printf("=====\n");
printf("Ви ввели %d фільмів.\n", ListItemCount(&movies));
printf("=====\n");

// очищення списку
EmptyTheList(&movies);
printf("Програма завершена.\n");
return 0;
}

void showmovies(Item item)
{
    printf("Рейтинг: %3d.  Фільм: %s\n",
           item.rating, item.title);
}
```



# Реалізація програми

```
char *s_gets(char *st, int n)
{
    char *ret_val;
    char *find;

    ret_val = fgets(st, n, stdin);
    if(ret_val)
    {
        find = strchr(st, '\n');
        if(find)
            *find = '\0';
        else
            while(getchar() != '\n')
                continue;
    }
    return ret_val;
}
```



# Реалізація інтерфейсу

```
//=====
// list.c - файл для функцій підтримки операцій зі списком
//=====
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

// прототип локальної функції
static void CopyToNode(Item item, Node *pnode);

//=====
// Функції інтерфейсу
//=====
// Встановлює список в пустий стан (ініціалізація)
//=====
void InitializeList(List *plist)
{
    *plist = NULL;
}
```



# Реалізація інтерфейсу

```
//=====
// Повертає true, якщо список пустий
//=====
bool ListIsEmpty(const List *plist) {
    if(*plist == NULL) return true;
    else return false;
}
//=====
// Повертає true, якщо список повний
//=====
bool ListIsFull(const List *plist) {
    Node *pt;
    bool full;
    pt = (Node *) malloc(sizeof(Node));
    if(pt == NULL) full = true;
    else full = false;
    free(pt);
    return full;
}
```





# Реалізація інтерфейсу

```
//=====
// Повертає кількість вузлів
//=====
unsigned int ListItemCount(const List *plist)
{
    unsigned int count = 0;
    Node *pnode = *plist;           // встановлюємо вказівник
                                    // на початок списку

    while (pnode != NULL)
    {
        ++count;
        pnode = pnode->next;        // встановлюємо вказівник
                                    // на наступний вузол
    }
    return count;
}
```



# Реалізація інтерфейсу

```
//=====
// Створює вузол для зберігання елемента і додає його в кінець
// списку, на який вказує змінна plist (повільна реалізація)
//=====
bool AddItem(Item item, List *plist)
{
    Node *pnew;
    Node *scan = *plist;
    pnew = (Node *) malloc(sizeof(Node));
    if(pnew == NULL)
        return false;    // вихід з функції у разі помилки
    CopyToNode(item, pnew);
    pnew->next = NULL;
    if(scan == NULL)    // список пустий, тому треба помістити
        *plist = pnew; // pnew на початок списку
    else
    {
        while(scan->next != NULL)
            scan = scan->next;    // пошук кінця списку
```



# Реалізація інтерфейсу

```
        scan->next = pnew;           // додавання pnew в кінець
    }
    return true;
}

//=====
// Відвідує кожен вузол і виконує функцію, на яку вказує pfun
//=====
void Traverse(const List *plist, void (*pfun) (Item item))
{
    Node *pnode = *plist;  // встановлюємо на початок списку
    while (pnode != NULL)
    {
        (*pfun) (pnode->item);  // застосовуємо функцію до
                                // елементу
        pnode = pnode->next;    // перехід до наступного
                                // елемента
    }
}
```

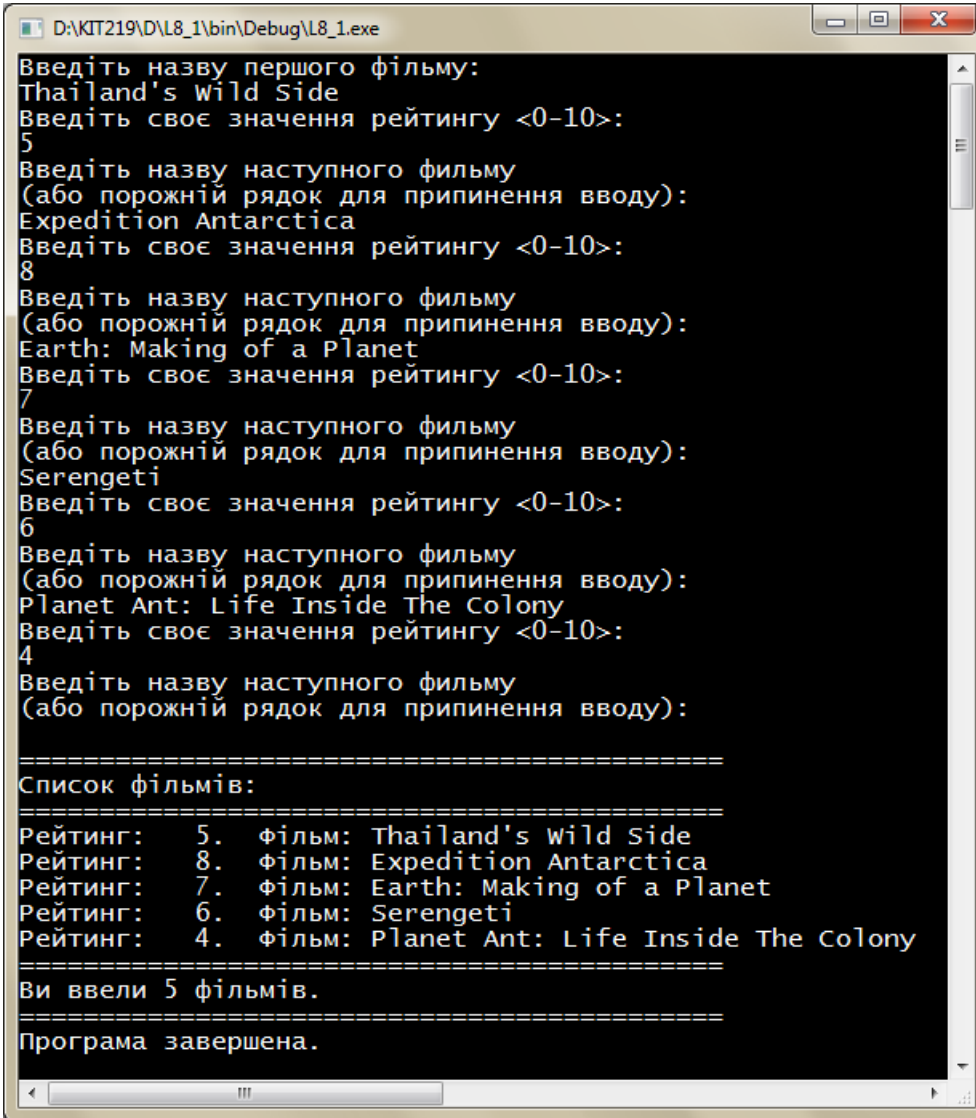


# Реалізація інтерфейсу

```
//=====
// Звільняє пам'ять, яка була виділена функцією malloc()
// Встановлює вказівник списку в NULL
//=====
void EmptyTheList(List *plist) {
    Node *psave;
    while(*plist != NULL) {
        psave = (*plist)->next; // зберігання адреси
                                // наступного вузла
        free(*plist);           // звільнення поточного вузла
        *plist = psave;         // перехід до наступного вузла
    }
}

//=====
// Визначення локальної функції. Копіює елемент у вузол
//=====
static void CopyToNode(Item item, Node *pnode) {
    pnode->item = item;          // копіювання структури
}
```

# Реалізація інтерфейсу



```
D:\KIT219\D\L8_1\bin\Debug\L8_1.exe
Введіть назву першого фільму:
Thailand's wild Side
Введіть своє значення рейтингу <0-10>:
5
Введіть назву наступного фільму
(або порожній рядок для припинення вводу):
Expedition Antarctica
Введіть своє значення рейтингу <0-10>:
8
Введіть назву наступного фільму
(або порожній рядок для припинення вводу):
Earth: Making of a Planet
Введіть своє значення рейтингу <0-10>:
7
Введіть назву наступного фільму
(або порожній рядок для припинення вводу):
Serengeti
Введіть своє значення рейтингу <0-10>:
6
Введіть назву наступного фільму
(або порожній рядок для припинення вводу):
Planet Ant: Life Inside The Colony
Введіть своє значення рейтингу <0-10>:
4
Введіть назву наступного фільму
(або порожній рядок для припинення вводу):

=====
Список фільмів:
=====
Рейтинг: 5. Фільм: Thailand's wild Side
Рейтинг: 8. Фільм: Expedition Antarctica
Рейтинг: 7. Фільм: Earth: Making of a Planet
Рейтинг: 6. Фільм: Serengeti
Рейтинг: 4. Фільм: Planet Ant: Life Inside The Colony
=====
Ви ввели 5 фільмів.
=====
Програма завершена.
```