МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”

ДИНАМІЧНІ СТРУКТУРИ ДАНИХ

ІНСТРУКЦІЯ
до лабораторної роботи № 10 з курсу
“Основи програмування”
для базового напрямку “Програмна інженерія”

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – оволодіти практичними прийомами створення та опрацювання динамічних списків.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Загальне поняття про динамічні структури

Динамічні структури даних характеризуються відсутністю фізичної суміжності елементів структури в пам'яті (різні елементи «розкидані» в пам'яті), а також непостійністю та непередбачуваністю розміру (в процесі виконання програми можуть додаватися нові елементи чи видалятися існуючі). Динамічні структури являють собою сукупність об’єктів (як правило, розміщених у динамічній пам'яті), кожен з яких містить інформацію про адресу в пам'яті іншого об’єкта, зв’язаного з ним.

Оскільки елементи динамічної структури розміщуються за непередбачуваними адресами пам'яті, то адресу кожного елемента структури неможливо обчислити за адресою попереднього або наступного елемента. Між елементами динамічної структури встановлюються явні зв’язки за допомогою вказівників. Таке представлення даних у пам'яті називається зв’язним. Елемент динамічної структури складається з двох полів:

- інформаційного поля (поля даних), яке містить ті дані, для яких власне і створюється структура;
- поле зв'язків, в якому містяться один або декілька вказівників, що зв’язують даний елемент з іншими елементами структури.

Інформаційне поле може бути представлене не лише сукупністю даних вбудованих типів, але й масивами, структурами тощо.

Розмір динамічної структури обмежується лише доступним обсягом машинної пам'яті. При зміні логічної послідовності елементів структури вимагається не переміщення даних у пам'яті, а лише корекція вказівників.

Однак, робота зі вказівниками вимагає більшої кваліфікації програміста, для зберігання вказівників витрачається додаткова пам'ять, а доступ до елементів зв’язної структури може бути менш ефективним за часом. Для порівняння: при суміжному розміщенні однорідних даних у пам'яті (яке має місце в масиві) для обчислення адреси будь-якого елемента потрібно знати номер цього елемента та адреси початку області пам'яті (доступ прямий), а для доступу до елемента динамічної структури потрібно послідовно «перебрати» усі елементи динамічної структури від початку.

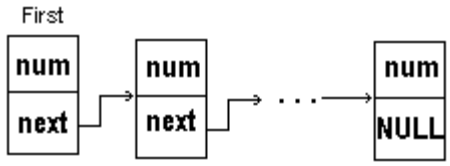
2.2 Списки

Список – це впорядкована множина, що складається зі змінного числа елементів, до яких застосовні операції додавання та видалення елементів. Якщо вказівник посилається лише на один інший елемент списку, то такий список називається однонапрямленим.

Наприклад, елемент однонапрямленого списку може описуватися структурою:

```
struct Item {
    int num;
    struct Item* next;
};
```

Відповідно, список схематично виглядатиме так:



First – це перший елемент списку («голова списку»), а останній елемент списку ні на що не посилається – його поле next встановлене у NULL.

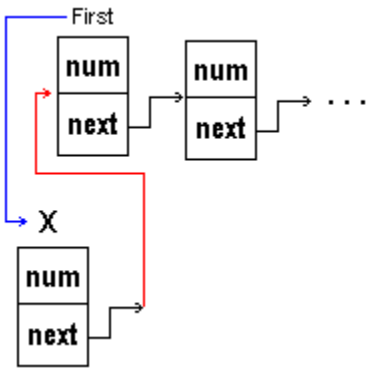
Елемент списку виділяється в динамічній пам'яті:

```
struct Item *p = (struct Item*)malloc(sizeof(struct Item));
```

Для додавання нового елемента на початок списку слід створити новий елемент X, в його поле next записати вказівник на елемент списку, що в поточний момент є першим, і вказівникові, який позначає перший елемент списку, присвоїти значення X. Наприклад:

```
struct Item *p = (struct Item*)malloc(sizeof(struct Item));
p->next = First;
First = p; /* First є вказівником типу Item, припускаємо, що в попередніх ділянках коду вказівник First посилався на перший елемент списку */
```

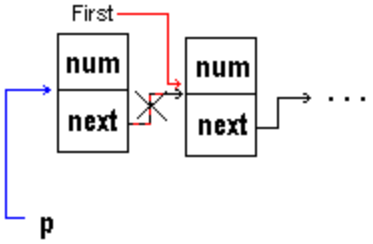
Схематично це можна зобразити так:



Видалення першого елемента списку можна здійснити так:

```
struct Item *p = (struct Item*)malloc(sizeof(struct Item));
p = First;
First = First->next; /* тепер першим буде той елемент, який раніше був другим */
free(p); /* звільняється пам'ять, яку займав перший елемент */
```

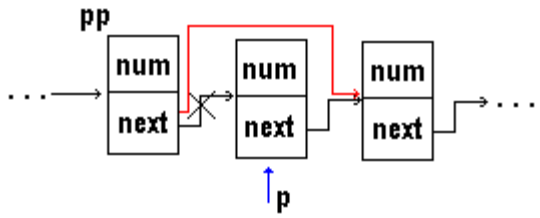
Схематично процес видалення першого елемента можна зобразити так:




```
struct Item *p = (struct Item*)malloc(sizeof(struct Item));
```

```
    p = pp; p = p-next; /* нехай pp – вже визначений раніше вказівник на існуючий елемент списку */
    pp->next = p->next; /* тепер вказівник pp посилається на елемент «через 1» */
```

Схематично видалення елемента виглядає так:



Якщо елемент посилається і на попередній, і на наступний елемент, то такий список є двонапрямленим.

Якщо вказівник в останньому елементі не встановлений у NULL, а посилається на перший елемент списку, то такий список називається кільцевим.

Приклад. В наступній програмі зчитується довільна кількість цілих чисел, поки користувач не задасть число 0. Усі зчитані числа записуються в однонапрямлений список, який згодом роздруковується.

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    struct Item {
        int num; /* інформаційне поле елементів списку представлене лише одним цілим числом */
        struct Item* next; /* для зв'язування елементів списку використовується вказівник на структуру Item */
    };

    struct Item *p = (struct Item*)malloc(sizeof(struct Item));
    struct Item *start = NULL; /* start служитиме вказівником на початок списку; задаємо початкове значення NULL */

    int n;
    do {
        scanf("%d", &n);
        p->num = n; /* зчитане ціле число записуємо в поле num першого елемента списку */
        if (start->next == NULL) /* для першого поле next елемента start ще є рівним NULL, а тому  в нього буде записано адресу першого елемента; це присвоєння виконається лише один раз */
        {
            start->next = p;
            start->num = p->num;
        }
        struct Item *pp = (struct Item*)malloc(sizeof(struct Item)); /* створюємо новий елемент списку 0 */

        p->next = pp; /* з поточного елемента посилаємося на щойно створений елемент, і робимо повторне присвоєння – на наступній ітерації циклу введене число буде записане у поле щойно створеного елемента списку */
```

```
          p = pp;
     } while (n); /* читання даних з клавіатури відбуватиметься, поки не буде введено 0
*/

     p->next = NULL; /* після виходу з циклу записуємо NULL у поле next останнього
опрацьованого елемента списку */

     p = start->next; /* виводимо елементи списку на екран */
     while (p->next != NULL) {
          printf("%d\t", p->num);
          p = p->next;
     };
     _getch();
}

/* Інший варіант */

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>


int main()
{
     struct Item {
          int num;
          struct Item* next;
     };

     struct Item *p = (struct Item*)malloc(sizeof(struct Item));
     struct Item *start =NULL; /* тут одразу вказівник start рівний нулю */

     int n;
     do {
          scanf("%d", &n);
          p->num = n;
          if (!start)
               start = p;
          struct Item *pp = (struct Item*)malloc(sizeof(struct Item));
          p->next = pp;
          p = pp;
     } while (n);
     p->next = NULL;
     p = start;
     while (p->next != NULL) {
          printf("%d\t", p->num);
          p = p->next;
     };
     _getch();
}
```

Приклад. В наступній програмі зчитується у нескінченному циклі довільна кількість рядків символів, довжина кожного з яких не перевищує 100 символів. Якщо користувач ввів слово “end”, відбувається вихід з циклу. Усі рядки записуються в однонапрямлений список, який згодом виводиться на екран.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main()
{
	struct Item {
		char str[100];
		struct Item* next;
	};

	struct Item *p = (struct Item*)malloc(sizeof(struct Item));
	struct Item *start = NULL;
	char str[100];
	while (1) {
		if (!start) start = p;
		scanf("%s", str);
		if (!strcmp(str, "end")) break;
		strcpy(p->str, str);
		struct Item *pp = (struct Item*)malloc(sizeof(struct Item));
		p->next = pp;
		p = pp;
	};
	p->next = NULL;
	p = start;
	while (p->next != NULL) {
		printf("%s\n", p->str);
		p = p->next;
	};
	_getch();
}
```

Приклад. В наступній програмі з клавіатури задаються цілі числа, як додатні, так і від’ємні, в довільній кількості, поки не буде введено 0. Числа записуються в однонапрямлений список, який виводиться на екран. Далі з цього списку вилучаються всі від’ємні числа. Результат виводиться на екран.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main()
{
	struct Item {
		int num;
		struct Item* next;
	};
	struct Item *p = (struct Item*)malloc(sizeof(struct Item));
	struct Item *start = NULL;
	int n;
```

```
     do {
          scanf("%d", &n);
          p->num = n;
          if (!start) start = p;

          struct Item *pp = (struct Item*)malloc(sizeof(struct Item));
          p->next = pp;
          p = pp;
     } while (n);
     p->next = NULL;
     p = start;
     while (p->next != NULL) {
          printf("%d\t", p->num);
          p = p->next;
     };
     p = start; /* встановлюємо вказівник p на початок списку */
     while (1) {
          struct Item* pp = p; /* у допоміжний вказівник pp запам'ятовуємо поточний
вказівник p і переходимо до наступного елемента списку; таким чином вказівник pp "тримає"
елемент-попередник */
          p = p->next;
          if (!p) break; /* при досягненні кінця списку вихід з циклу */
          if (p->num < 0) {/* якщо аналізований елемент списку містить від'ємне число, то
його треба видалити, а попередній елемент списку пов'язати з наступним за аналізованим
елементом */
               if (!p->next) {/* якщо ж для даного елемента не існує наступного елемента
списку */
                    pp->next = NULL; /* то вказуємо, що попередній елемент списку стає
останнім елементом, і виходимо з циклу */
                    break;
               }
               pp->next = p->next; /* якщо ж наступний елемент існує, то зв'язуємо з ним
попередній елемент, а поточний елемент p видаляємо */
               free(p);
               p = pp; /* вказівник p використаємо для організації наступної ітерації
циклу */
          }
     };
     /* Однак, при наведеній організації коду перший елемент ніколи не буде видалений,
оскільки аналіз елементів списку починається з наступного за першим, тобто, з другого
елемента списку. Тому потрібно окремо розглянути перший елемент списку */
     p = start;
     if (p->num < 0) {/* Якщо він містить від'ємне число, то список почнемо з наступного
елемента, якщо такий існує, а інакше список буде пустим */
          if (p->next)
               start = p->next;
          else start = NULL;
     }
     /* Повторний вивід списку після видалення від'ємних елементів */
     p = start;
     printf(" results: \n");
     while (p) {
          printf("%d\t", p->num);
          p = p->next;
     };
     _getch();
}
```

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Як формуються динамічні структури?
2. Де зберігаються динамічні дані?
3. Які переваги динамічних структур даних над статичними?
4. Що таке динамічний список, яка його структура?
5. Як виглядає однозв'язний динамічний список?
6. Які операції можуть використовуватися для опрацювання однозв’зного списку?
7. Що таке кільцевий список?
8. Яку структуру має двозв’язний список?
9. Які переваги двозв’язного списку перед однозв’язним?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання.
3. Скласти програму на мові С++ у відповідності з розробленим алгоритмом.
4. Виконати обчислення по програмі.
5. Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования С. - М. - Финансы и статистика. - 1992. – 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык С. Руководство для начинающих. - М. - Мир. - 1988. –512 с.
3. К. Джамса. Учимся программировать на языке С++. М.: Мир, 1997. – 320 с
4. Герберт Шилдт. Полный справочник по С++. М. – С.-П.-К., Вильямс. – 2003. – 800 с.
5. Демидович Е. М. Основы алгоритмизации и программирования. Язык Си. (Учебное пособие). – Санкт-Петербург: “БХВ Петербург”. – 2006. – 439 с.

6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Виконати завдання з лабораторної роботи № 11, організувавши послідовність структур в однозв’язний список. Реалізувати операцію вставки нового елемента у відсортований список і операцію вилучення зі списку даних, які відповідають одній з наступних умов:

1) про студентів з рейтинговим балом нижчим середнього в групі;
2) про студентів з оцінками 2;
3) про студентів, які не мають оцінки 5;
4) про студентів з рейтинговим балом нижчим 4,5;
5) про трьох студентів з найнижчим рейтинговим балом;
6) про студентів з рейтинговим балом нижчим 3,5;
7) про студентів з двома оцінками 2;
8) про студентів, які не мають оцінок 4 і 5;
9) про студентів, які отримали на другому іспиті оцінку 3;
10) про студентів, які не мають оцінки 2;
11) про студентів, які отримали на першому іспиті оцінку 2;
12) про студентів, які отримали на першому та третьому іспитах оцінки 3;
13) про студентів, які під час екзаменаційної сесії отримали оцінки 2, 3, 4, 5;
14) про студентів з рейтинговим балом вищим 4,5;
15) про книги з вартістю менше 5 грв.;

16) про книги з кількістю сторінок меншою 50 стор.;
17) про книги, видані раніше 1991 року;
18) про книги з назвою, що починається на букву Д;
19) про книги авторів з прізвищем, що починається на букву Я;
20) про книги з вартістю меншою від середньої у бібліотеці;
21) про книги з кількістю сторінок меншою від середньої у бібліотеці;
22) про книги видані раніше 1980 року з кількістю сторінок меншою 90;
23) про книги з вартістю більшою від середньої з кількістю сторінок меншою від середньої у бібліотеці;
24) про книги видані за останні 5 років;
25) про книги з назвою, що починається на букви П, К, Л;
26) про книги авторів, прізвища яких починаються на букви А, Б, В, Г;
27) про книги видані раніше 2000 року з кількістю сторінок меншою 150;