Міністерство освіти і науки України
Національний університет водного господарства та природокористування
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних технологій та економічної кібернетики

04-05-32

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з навчальної дисципліни ПРОГРАМУВАННЯ (Частина 3. Лінійні динамічні структури даних. Реалізація мовою програмування C#) для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інформаційні системи та технології» спеціальності 126 «Інформаційні системи та технології» та за освітньо-професійною програмою «Комп'ютерні технології» спеціальності 015 «Професійна освіта» денної та заочної форми навчання

Рекомендовано науково-методичною радою з якості ННІ АКОТ
Протокол № 10 від 22.06.2020 р.

Рівне – 2020

Методичні вказівки до виконання лабораторних робіт з навчальної дисципліни ПРОГРАМУВАННЯ (Частина 3. Лінійні динамічні структури даних. Реалізація мовою програмування C#) для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Інформаційні системи та технології» спеціальності 126 «Інформаційні системи та технології» денної форми навчання [Електронне видання] / Шевченко І. М. – Рівне : НУВГП, 2020. – 107 с.

Укладач: Шевченко І. М., старший викладач кафедри комп'ютерних технологій та економічної кібернетики.

Відповідальний за випуск: Грицюк П. М., д.е.н., професор, завідувач кафедри комп'ютерних технологій та економічної кібернетики.

Керівник групи забезпечення
спеціальності Гладка О. М.

Зміст

Лабораторна робота № 1. Поняття структури даних. Класифікація структур даних у програмах користувача та у пам‘яті комп‘ютера. Адресація, статична і динамічна пам'ять. Вказівники у мові програмування C#. Робота з областями динамічної пам'яті: виділення, обробка, вивільнення.

Мета роботи: набути навичок керування динамічною пам’яттю за допомогою використання вказівників.

Послідовність виконання роботи:

1. Ознайомитись із теоретичними відомостями. (*Актуалізація опорних знань*).
2. Виконати програмування програм за поданими прикладами. Результат представити викладачу (*Застосування набутих знань*).
3. Виконати варіант самостійної роботи. (*Закріплення набутих знань*).
4. Оформити звіт на виконану роботу. (*Узагальнення та систематизація набутих знань*).
5. Захист звітів, відповіді на запитання.

1.1. Теоретичні відомості

Поняття структури даних. Будь-яка програма, призначена для реалізації на ЕОМ, являє собою формальний опис алгоритму вирішення тієї чи іншої задачі. Дії, що виконуються програмою відповідно до цього алгоритмом, спрямовані на перетворення деяких об'єктів, що визначають поточний стан процесу рішення задачі. Такі внутрішні об'єкти програми будемо називати даними.

Основна мета будь-якої програми полягає в обробці даних. Дані різних типів зберігаються в пам'яті комп'ютера і обробляються по-різному. Згідно з концепцією типів даних, кожен тип даних однозначно визначає: множину значень, які

може приймати змінна заданого типу; операції та функції, які можна застосовувати до цієї змінної; внутрішнє подання змінної в пам'яті комп'ютера. При цьому пам'ять виділяється не для типу даних, а для розміщення змінних або констант певних типів. Основні типи даних, що застосовуються в програмуванні, представлені на рисунку 1.1.

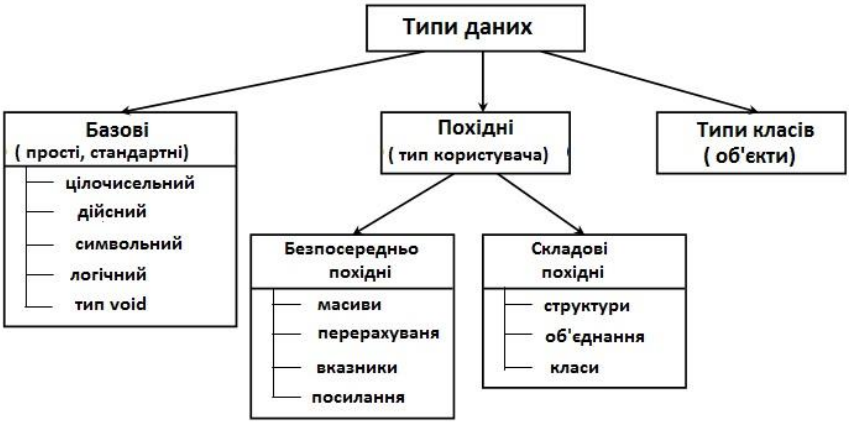


Рис. 1.1. Основні типи даних, що застосовуються в програмуванні

Вказівка типу даних чітко визначає:

- розмір пам'яті, що відводиться під даний програмний об’єкт і спосіб його розміщення в пам'яті;
- значення, допустимі для даного типу даних;
- операції, які можливо над цими даними виконувати.

Будемо виходити з того уявлення, що структура даних – програмна одиниця, що дозволяє зберігати й обробляти безліч однотипних або логічно пов'язаних даних в обчислювальній техніці. Вона формується за допомогою типів даних, посилань і операцій над ними в обраній мові програмування. Структура даних відноситься за своєю суттю до „просторових” понять: її можна звести до схеми організації інформації в пам’яті комп’ютера. Алгоритм же є відповідним процедурним елементом в структурі програми – він служить рецептом розрахунку. Структури даних, які застосовуються в алгоритмах,

можуть бути досить складними. Вибір правильного представлення даних часто служить ключем до вдалого програмування і може в більшій мірі впливати на продуктивність програми, ніж деталі реалізації використовуваного алгоритму. Але, мабуть, ніколи не появиться загальна теорія вибору структур даних, у кожному конкретному випадку потрібно підходити до цього творчо.

Незалежно від змісту і складності будь-які дані в пам'яті комп'ютера представляються послідовністю бінарних розрядів, а їх значеннями є відповідні бінарні числа. Дані, які розглядаються у вигляді послідовності бітів, мають дуже просту організацію, тобто є слабо структурованими. Більш крупні й змістовніші, ніж біт, „будівельні блоки" для організації довільних даних отримуються на основі поняття „структури даних".

Поняття „фізична структура даних" відображає спосіб фізичного представлення даних в пам'яті машини і називається ще структурою зберігання.

Розгляд структури даних без врахування її представлення в машинній пам'яті називається абстрактною або логічною структурою. В загальному випадку між логічною і відповідною їй фізичною структурами існує відмінність, міра якої залежить від самої структури і особливостей того середовища, в якому вона повинна бути відображена.

Будь-яка структура даних може описуватися, таким чином, на трьох різних рівнях:

- функціональна специфікація – вказує для деякого класу імен операції, які дозволені з цими іменами, і властивості цих операцій;
- логічний опис – задає декомпозицію об'єктів на більш елементарні об'єкти і декомпозицію відповідних операцій на більш елементарні операції;
- фізичне представлення – дає метод розміщення в пам'яті комп'ютера тих величин, які складають структуру, і

відношення між ними, а також спосіб кодування операцій на мові програмування.

Одній і тій же функціональній специфікації можуть відповідати декілька логічних описів, які в свою чергу можуть реалізовуватися декількома фізичними представленнями. Проте, потрібно мати впевненість, що декомпозиція кожного нового рівня достатньо добре відображає декомпозицію безпосередньо вищого рівня.

Найпростіші структури даних, реалізовані мовою програмування, називають також стандартними типами даних. Багато мов програмування дозволяють на основі стандартних типів будувати типи даних, визначені програмістом (користувачем).

Робота з великими наборами даних автоматизується простіше, коли дані упорядковані, тобто утворюють задану структуру.

Розрізняють фізичну та логічну структуру даних. Фізична структура, на відміну від логічної, визначає спосіб представлення даних в пам'яті комп'ютера.

Розрізняють прості структури данных (типи) та складні (інтегровані). Прості структури не можуть бути поділені на складові частини, більші ніж біти. З точки зору фізичної структури для простого типу даних чітко визначений його розмір та спосіб розміщення в оперативній пам'яті комп'ютера. З точки зору логічної структури прості дані є неподільними одиницями.

Інтегровані структури даних включають у себе інші структури даних прості чи інтегровані. Між окремими елементами структури можуть існувати явно задані зв'язки (не обов'язково). В залежності від цього розрізняють:

незв'язані структури (вектори, матриці, рядки, стеки, черги) та зв'язані структури (звязані списки).

Класифікація структур даних у програмах користувача та у пам'яті комп'ютера. За ознакою можливості зміни розміру розрізняють структури статичні, напівстатичні та динамічні. Під

зміною розуміють зміну числа елементів структури або зв’язків між її елементами. Класифікація структур даних за ознакою зміни наведена на рисунку 1.2.

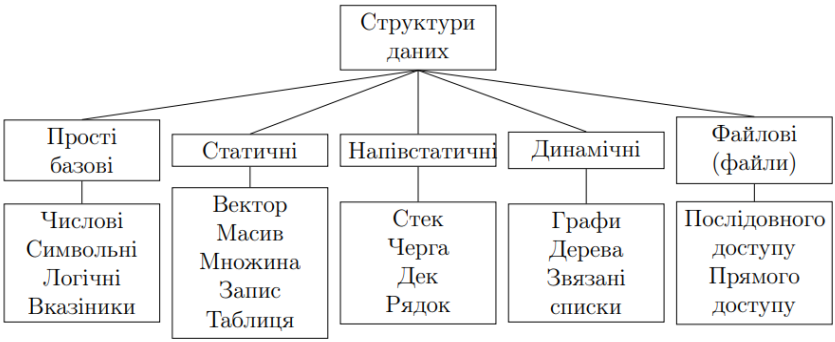


Рис. 1.2. Класифікація структур даних.

За ознакою впорядкованості елементів структури можна поділити на лінійні та нелінійні. Прикладом нелінійних структур можуть бути багатозв’язні списки, дерева, графи. Лінійні структури за характером розподілу елементів в оперативній пам’яті, в свою чергу, поділяються на структури із послідовним розподілом (вектори, рядки, масиви, стеки, черги) та структури із довільними розподілом (однозв’язні, двохзв’язні списки).

Адресація, статична і динамічна пам'ять. Статичні змінні характеризуються тим, що їх значення зберігаються в ділянках оперативної пам'яті, які визначаються на етапі компіляції програми і не змінюються під час її виконання. Проте у багатьох задачах обсяг оперативної пам'яті, необхідної для збереження певних даних, неможливо визначити наперед. Для зберження таких даних використовуються змінні, які створюються і знищуються в процесі виконання програми. Такі змінні називаються *динамічними*, а пам'ять, що для них виділяється, — *динамічною пам'яттю.* Оскільки обсяг оперативної пам'яті, що використовується для збереження значення динамічної змінної, компілятору не відомий, він не позначає динамічну змінну ідентифікатором. Доступ до значення такої змінної здійснюється

за її *адресою.* Отже, на етапі компіляції програми виділяється оперативна пам'ять для збереження адреси динамічної змінної, а пам'ять для збереження її значення виділяється під час виконання програми.

Оперативна пам’ять комп'ютера є послідовністю байтів, або *комірок.* Розташування таких комірок є впорядкованим, і тому їх можна пронумерувати. Послідовна нумерація байтів цілими числами є зручною з погляду людини, проте процесор використовує інший спосіб доступу до комірок пам'яті – доступ за допомогою *адрес.*

Адреса складається з двох шістнадцятирозрядних чисел, що називаються базисом сегмента та зсуненням. *Сегмент* — це неперервна область оперативної пам'яті обсягом 64 Кбайт (65 536 байт), що починається з комірки, номер якої є кратним 16. *Базис сегмента* – це номер 16-байтової групи, з якої починається сегмент. Таким чином, якщо базис сегмента дорівнює *x,* то цей сегмент починається з комірки, що має номер *16x. Зсунення* дорівнює відстані в байтах, на яку комірка віддалена від початку сегмента. В адресі комірки базис сегмента та зсунення записуються, як правило, у шістнадцятковому вигляді і розділяються символом «:». Наприклад, 000A:001A – це адреса комірки, номер якої дорівнює (A16-1016) + 1A16= BA16= 18610. Зазначимо, що адреси можна зіставляти не лише з окремими комірками, а і з довільними неперервними ділянками пам'яті. Адресою ділянки пам'яті вважається адреса її найпершої комірки.

Пам’ять комп’ютера, яку системи програмування виділяють для роботи програм, можна поділити на 4 частини:

- сегмент програмного коду;
- сегмент даних;
- сегмент програмного стеку;
- динамічна пам'ять, що використовується для збереження значень динамічних змінних.

Пам’ять під програмний код призначена для збереження програм у внутрішньому представленні системи програмування або у машинних кодах.

Пам’ять під дані призначена для збереження статичних даних програм (глобальних змінних та констант).

Програмний стек призначений для тимчасового збереження локальних змінних, параметрів підпрограм, організації виклику підпрограм та обчислення виразів.

Купа призначена для динамічного розподілу пам’яті під структури даних.

Пам’ять під програмний код та дані є статичною, а програмний стек та купа – динамічною пам’яттю. Програмним стеком керує система програмування без втручання програміста в той час, як динамічна пам’ять спеціально призначена для використання програмістом. Отже, динамічна пам'ять є неперервним масивом байтів. Ця область пам'яті називається *купою*, або Heap-областю (від англ. Heap – купа).

Вказівники у мові програмування C#. Одним з основних переваг мови C # є його схема роботи з пам'яттю: автоматичне виділення пам'яті під об’єкти і автоматична прибирання сміття. При цьому неможливо звернутися за неіснуючою адресою пам'яті або вийти за межі масиву, що робить програми більш надійними і безпечними і виключає можливість появи цілого класу помилок, що доставляють масу незручностей при написанні програм на інших мовах.

Однак в деяких випадках виникає необхідність працювати з адресами пам'яті безпосередньо, наприклад, при взаємодії з операційною системою, написанні драйверів або програм, час виконання яких критично. Таку можливість надає так званий небезпечний (unsafe) код.

Незахищеним називається код, виконання якого середовище CLR не контролює. Він працює безпосередньо з адресами областей пам'яті за допомогою вказівників і повинен бути явним чином позначений за допомогою ключового слова *unsafe,* яке визначає так званий небезпечний контекст виконання.

Ключове слово *unsafe* може використовуватися або як специфікатор, або як оператор. У першому випадку його вказують поряд з іншими специфікаторами при описі класу, делегата, структури, методу, поля і т. д. – скрізь, де допустимі інші специфікатори. Це визначає небезпечний контекст для описуваного елементу, наприклад:

```
public unsafe struct Node
        {
                public int Value;
                public Node* Left;
                public Node* Right;
}
```

Оператор unsafe має такий синтаксис:

unsafe блок операторів

Всі оператори, що входять в блок, виконуються в небезпечному контексті.

Компіляція коду, що містить небезпечні фрагменти, повинна проводитися з ключем /unsafe. Цей режим можна встановити шляхом настройки середовища Visual Studio: **Проект–>Свойства–>Сборка–>Разрешить небезопасний код (Project–> Properties–> Build–> Allow Unsafe Code)** (рис. 1.3):

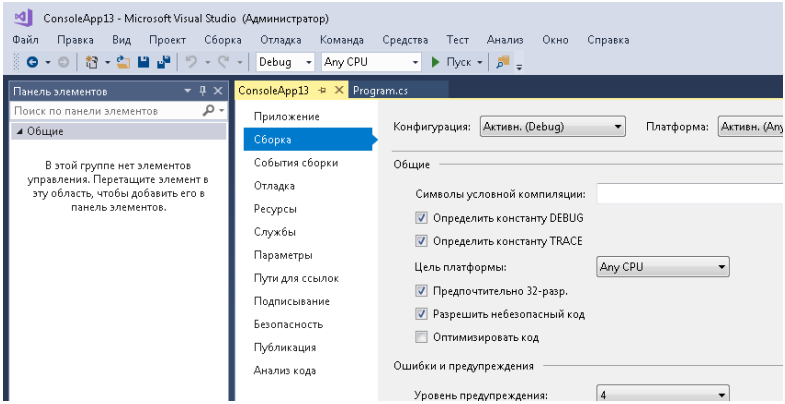


Рис 1.3. Встановлення небезопасного режиму

Не можна оголосити локальну змінну як unsafe. Якщо така потреба виникає, то цю змінну потрібно розмістити всередині незахищеного блоку.

Для роботи з динамічною пам’яттю використовують вказівники. Вказівники призначені для зберігання адрес областей пам'яті. Синтаксис оголошення вказівника такий:

тип даних * змінна;

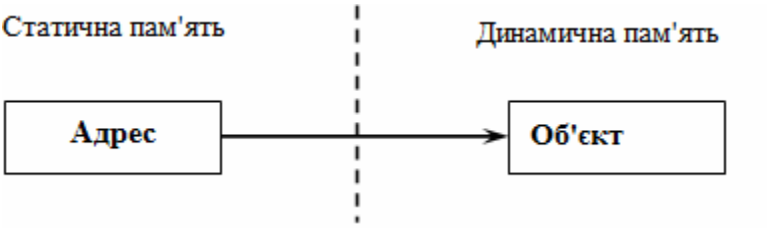


Рис. 1.4. Зв’язок покажчика з адресованим об’єктом

Покажчик містить адресу певного об’єкта в динамічній пам'яті. Сам покажчик є статичним об’єктом і розташований у сегменті даних (рис. 1.4).

Тип даних в цьому оголошенні не може бути класом, але може бути структурою, переліком, вказівником, або одним зі стандартних типів: sbyte, byte, short, ushort, int, uint, long,ulong, char, float, double, decimal, bool, void.

Приклади оголошення вказівників:

```
int* a;             //  вказівник на int
Node* pNode;        //  вказівник на структуру Node
void* p;            //  вказівник на невизначений тип
int*[] m;           // одновимірний масив  вказівників на int
int** d;            //  вказівник на  вказівник на int
```

В одному операторі можна описати кілька вказівників одного і того ж типу, наприклад:

```
int* a, b, c;       // три  вказівника на int
```

Вказівники є *окремою категорією* типів даних. Вони не успадковуються від типу *object*, і перетворення між типом *object* і типами вказівників неможливо. Зокрема, для них не

виконується упаковка і розпакування. Однак допускаються перетворення між різними типами вказівників, а так само вказівниками і цілими.

Вказівники можуть бути локальними змінними, полями, параметрами і повертаються значеннями функції. Ці величини підкоряються загальним правилам визначення області дії і часу життя.

Вказівник типу *void* означає, що він посилається на змінну невідомого типу. Вказівник на тип *void* застосовується в тих випадках, коли конкретний тип об'єкта, адреса якого потрібно зберігати, не визначений (наприклад, якщо в одній і тій же змінної в різні моменти часу потрібно зберігати адреси об'єктів різних типів).

Вказівникові на тип *void* можна надати значення вказівника будь-якого типу, а також порівняти його з будь-якими вказівниками, але перед виконанням будь-яких дій з областю пам'яті, на яку він посилається, потрібно перетворити його до конкретного типу явно. Отже, вказівник *void* може перебувати тільки в лівій частині оператора присвоювання.

Для вказівників підтримуються *неявні перетворення з будь-якого типу вказівника до типу void**. Будь-якому вказівнику можна надати константу *null* (невизначена адреса). Крім того, допускаються явні перетворення:

- між вказівниками будь-якого типу;
- між вказівниками будь-якого типу і цілими типами (зі знаками і без знака).

Коректність перетворень лежить на сумлінні програміста. Перетворення ніяк не впливають на величини, на які посилаються вказівники, але при спробі отримання значення за вказівником невідповідного типу поведінка програми не визначено.

Робота з областями динамічної пам'яті: виділення, обробка, вивільнення. Ініціалізація вказівників може бути здійснена різними способами.

1 спосіб. Присвоєння вказівнику адреси існуючого об'єкта:

– *За допомогою операції одержання адреси:*

```
int a = 5;                          // ціла змінна
int * p = &a;                 // в  вказівник записується адреса а
```

– *За допомогою значення іншого вказівника:*

```
int * p 1 = p;   //припускаємо, що  вказівник р вже має значення
```

– *За допомогою імені масиву, яке трактується як адреса:*

```
int [] b = new int [] {10, 20, 30, 40};     // масив
fixed (int *p2 = b) {...}; //присвоювання вказівникові p2 адреси
початку масиву
```

– `fixed (int *p2 = & b [0]) {...};  //теж саме`

2 спосіб. Присвоєння вказівнику адреси області пам'яті в явному вигляді:

```
char * p3 = (char *) 0x12F69E;
```

Тут 0x12F69E – шістнадцяткова константа, (char *) – операція явного приведення типу: константа перетвориться до типу вказівника на char.

3 спосіб. Присвоєння нульового значення:

```
int *p4 = null;
```

4 спосіб. Виділення області пам'яті в стеку і присвоювання її адреси вказівником:

```
int * p5 = stackalloc int [10];
```

Тут операція *stackalloc* виконує виділення пам'яті під 10 величин типу int (масив з 10 елементів) і записує адресу початку цієї області пам’яті в змінну p5, яка може трактуватися як ім’я масиву, так і вказівником.

Основні операції з вказівниками наведені в таблиці 1.1.

Таблиця 1.1.

Основні операції з вказівниками

Операція	Опис
*	Розіменування – отримання значення, яке знаходиться за адресою, що зберігається у вказівникові.
->	Доступ до елементу структури через

	вказівник
[]	Доступ до елементу масиву через вказівник
&	Отримання адреси змінної
++, --	Збільшення і зменшення значення вказівника на один адресованих елементів
+, -	Додавання з цілої величиною і віднімання вказівників
==,*!*=, <>, <=,>=	Порівняння адрес, що зберігаються у вказівниках. Виконується як порівняння беззнакових цілих величин.
stackalloc	Виділення пам'яті в стеку під змінну, на яку посилається вказівник

Основними операціями з вказівниками є «*» (розіменування) і «&» (отримання адреси).

Обидві ці операції є унарними, тобто мають один операнд, перед якими вони ставляться. Операція «&» відповідає операції «взяти адресу». Операція «*» називається розадресацією або разіменуванням. Вона призначена для доступу до величини, адреса якої зберігається у вказівнику. Цю операцію можна використовувати як для отримання, так і для зміни значення величини.

Приклад:

```
int a = 5;                    // ціла змінна
int* p = &a;   // ініціалізація  вказівника адресою змінної a
Console.WriteLine(*p);   // операція разадресації, результат
дорівнює 5
Console.WriteLine(++(*p));   // результат 6
int[] b = new int[] { 10, 20, 30, 40 };   // створення масиву
fixed (int* t = b)   // ініціалізація  вказівника адресою
початку масиву
  {
    int* z = t;  // ініціалізація  вказівника значенням іншого
вказівника
    for (int i = 0; i < b.Length; ++i)
        {
           t[i] += 5;  // доступ до елементу масиву (збільшення
на 5)
```

```
            *z += 5;   // доступ за допомогою адресації
(збільшення ще на 5)
             ++z;       // інкремент  вказівника
        }
  }
```

У наведеному прикладі доступ до елементів масиву виконується двома способами: шляхом індексації вказівника t і шляхом разадресації вказівника *z*, значення якого инкрементируется при кожному проході циклу для переходу до наступного елементу масиву.

Конструкцію **змінна* можна використовувати *в лівій частині* оператора присвоювання, оскільки вона визначає адресу області пам'яті. Для простоти цю конструкцію можна вважати ім'ям змінної, на яку посилається вказівник. З нею допустимі всієї дії, визначені для величин відповідного типу.

Оператор *fixed* фіксує об'єкт, адреса якого заноситься у вказівник, для того щоб його не переміщував збирач сміття і, таким чином, вказівник залишився коректним. Фіксація відбувається на час виконання блоку, який записаний після круглих дужок.

Арифметичні операції з вказівниками (додавання з цілим, різниця вказівників, інкремент і декремент) автоматично враховують розмір типу величин, адресованих вказівниками. Ці операції застосовні тільки до вказівників *одного типу* і мають сенс в основному при роботі зі структурами даних, елементи яких розміщені в пам'яті послідовно, наприклад, з *масивами.*

Інкремент переміщує вказівник до наступного елементу масиву, декремент – до попереднього.

Різниця двох вказівників – це різниця їх значень, поділена на розмір типу в байтах. Так, якщо *p* і *p1* – вказівники на елементи одного і того ж масиву, то операція *p - p1* дає такий же результат, як і *віднімання індексів відповідних елементів масиву.*

До вказівників можна додавати ціле число. Нехай вказівник *p* має значення 2000 і вказує на тип *byte*. Тоді в результаті виконання оператора:

p = p +3;

значення вказівника *p* буде 2003. Якщо ж вказівник *p1 = 2000* був би вказівником па *float*, то після застосування оператора:

p1 = p1 + 10;

значення *p1* було б 2040.

Загальна формула для обчислення значення вказівника після виконання операції:

p = p + n;

матиме вигляд:

*<p> = <p> + n * <кількість байт пам'яті базового типу вказівника>*

Отже, при додаванні до вказівника цілого числа n фактично значення вказівника змінюється на величину ***n*sizeof(тип)*,** де *sizeof* – операція отримання розміру величини зазначеного типу (в байтах). Ця операція застосовується тільки в небезпечному контексті, з її допомогою можна отримувати розміри не тільки стандартних, але і призначених для користувача типів даних.

Інші арифметичні операції над вказівниками **заборонені,** наприклад не можна додати два вказівника, помножити вказівник на число і т. д.

Вказівники можна порівнювати. Можна застосувати всі шість операцій:<, >, <=, >=, ==,!=

Порівняння *p<g* означає, що *адреса,* що знаходиться в *p*, *менше адреси,* що знаходиться в *g*. Якщо *p* і *g* вказують на елементи одного масиву, то *індекс елемента,* на який вказує *p*, менше індексу масиву, на який вказує *g*.

Приклад застосування арифметичних операцій з вказівниками:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
```

```
    {
        unsafe static void Main(string[] args)
        {
            // демонстрація операції інкремента
            int* p;
            int a = 5;
            // вивід значення  вказівника (адреси),
             p = &a;
            // тип uint - беззнаковий цілий тип в unicode -
форматі (16- ве число)
            Console.WriteLine("{0}", (uint)p);
            // операція інкремента. Значення  вказівника
(адреса) //збільшується на 4  байта
            p++;
            Console.WriteLine("{0}", (uint)p);

            // аналогічні дії, тільки застосовні для типу long:
            long *p1;
            long a1 = 5;
            p1 = &a1;
            Console.WriteLine("{0}", (uint)p1);
            p1++;
            // значення  вказівника (адреса) збільшиться на 8
байт!
            Console.WriteLine("{0}", (uint)p1);

            // демонстрація операції різниці вказівників
            int[] a2 = new int[] { 1, 2, 3 };
            fixed (int *p2 = a2)
            {
                int* p3 = p2;
                p3++;
                Console.WriteLine("{0}", (uint)(p3 - p2));

                // демонстрація операції додавання  вказівника з
цілим числом
                int* p4 = p2 + 2; // до  вказівника додали 2
елементи!
                Console.WriteLine(*p4); // значення за
вказівником дорівнює 3
            }
            Console.ReadLine();
        }
    }
```

```
}
```

Записуючи вираз з вказівниками, слід звертати увагу на пріоритети операцій. Як приклад розглянемо послідовність дій, задану в операторі:

**P++=10;*

Оскільки інкремент постфіксний, він виконується після виконання операції присвоювання. Таким чином, спочатку за адресою, записаному в вказівнику *P*, буде записано значення 10, а потім вказівник збільшиться на кількість байтів, що відповідає його типу. Те ж саме можна записати докладніше:

**P = 10;*
P++;

Вираз *(*P)++*, навпаки, інкрементує значення, на яке посилається вказівник.

У наступному прикладі кожен байт беззнакового цілого числа *x* виводиться на консоль за допомогою вказівника *t*:

```
   uint x = 0xAB10234F;
   byte *t = (byte*)&x;
   for(int i = 0; i < 4; ++i)  Console.Write("{0:X}  ", *t++); //
результат 4F 23 10 AB
```

Попередньо вказівник *t* був встановлений на молодший байт змінної *x*.

У мові C# можлива також ситуація, коли *вказівник вказує на вказівник*. У цьому випадку опис буде мати такий вигляд: *int **point;*

Змінна *point* має тип *вказівник на вказівник на int*. Щоб отримати цілочисельне значення змінної, на яку вказує point слід використовувати: ***point;*

Приклад:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
```

```
    {
        unsafe static void Main(string[] args)
        {
            int i = 7;
            int* p;
            int** pp;
            p = &i;
            pp = &p;
            Console.WriteLine("i = {0:X} p = {1:X} pp = {2:X}",
i, (uint)p, (uint)pp);
            Console.WriteLine("i = {0}  *p = {1} **pp = {2}", i,
*p, **pp);
            ++*p;
            Console.WriteLine("i = {0} *p = {1} **pp = {2}", i,
*p, **pp);
            **pp = 12;
            Console.WriteLine("i = {0} *p = {1} **pp = {2}", i,
*p, **pp);
            Console.ReadLine();
        }
    }
}
```

Операція *stackalloc* дозволяє виділяти пам'ять в стеку під задану кількість величин заданого типу:

stackalloc тип[кількість]

Кількість задається цілочисельним виразом. Якщо пам'яті недостатньо, генерується виключення System.StackOverflowException. Виділена пам’ять нічим не ініціалізується і *автоматично звільняється* при завершенні блоку, що містить цю операцію. Приклад виділення пам’яті під п’ять елементів типу *int* і заповнення її числами від 0 до 4:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        unsafe static void Main(string[] args)
```

```
        {
            //Створення одного об'єкту типу int
            int *p = stackalloc int[sizeof(int)];
            *p = 5;
            Console.WriteLine(*p);           // Результат 1

            //Створення масиву з 5 об'єктів типу int
            int *p1 = stackalloc int[5];
            for (int i = 0; i < 5; ++i)
            {
                p1[i] = i;
                Console.Write(p1[i] + " ");           //
Результат 0 1 2 3 4
            }
            Console.ReadLine();
           }
    }
}
```

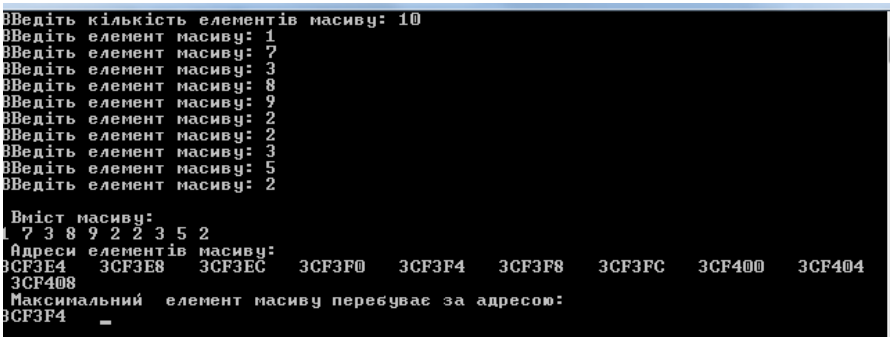
1.2. Приклади виконання завдань

Приклад 1.2.1. Написати програму для обробки масива даних з використанням арифметики вказівників. В масиві з 10 цілих чисел знайти адресу розташування найбільшого значення.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        unsafe static void Main(string[] args)
        {
            Console.Write("ВВедіть кількість елементів масиву:
");
            int N;
            N= Convert.ToInt32(Console.ReadLine());
            //Виділення місця для масиву
            int *p = stackalloc int[N];
            // Заповнення масиву значеннями
```

```
            for (int i = 0; i < N; ++i)
            {
                Console.Write("ВВедіть елемент масиву: ");
                *(p + i) = Convert.ToInt32(Console.ReadLine());
            }
            //Перегляд вмісту масиву
            Console.WriteLine("\n Вміст масиву: ");
            for (int i = 0; i < N; ++i)
            {
                Console.Write(*(p + i) + " ");
            }
            Console.WriteLine("\n Адреси елементів масиву: ");
            for (int i = 0; i < N; ++i)
            {
                Console.Write("{0:X}   ", (uint)(p+i)); ;
            }
            //Знаходження місця перебування найбільшого значення
масиву
            int max = *p;
            int* pmax=p;
            for (int i = 0; i < N; ++i)
            {
                if (max < *(p + i))
                {
                    pmax = p + i;
                    max = *(p + i);
                }
            }
            Console.WriteLine("\n Максимальний  елемент масиву
перебуває за адресою: ");
            Console.Write("{0:X}   ",(uint) pmax) ;
            Console.ReadLine();
        }
    }
}
```

Результат роботи програми:



1.3. Індивідуальні завдання

Потрібно реалізувати кожне завдання у відповідності з наведеними етапами:

1. вивчити словесну постановку задачі, виділяючи при цьому всі види даних;
2. сформулювати математичну постановку задачі;
3. обрати метод розв'язання задачі, якщо це необхідно;
4. розробити графічну схему алгоритму;
5. записати розроблений алгоритм мовою програмування;
6. розробити контрольний тест до програми;
7. налагодити програму;
8. представити звіт до роботи.

Варіант кожного завдання обирається за номером студента в журналі.

Завдання 1. Написати програму для обробки масива даних з використанням арифметики вказівників.

1. В масиві з 10 цілих чисел знайти адресу розташування найменшого значення.
2. Дано n дійсних чисел: $x_1, x_2, \ldots, x_n$. Знайти середнє арифметичне значення елементів масиву. Вивести на екран адреси елементів, значення яких більше середнього арифметичного значення.
3. Дано n дійсних чисел: $x_1, x_2, \ldots, x_n$. Знайти середнє геометричне значення елементів масиву. Вивести на екран

адреси елементів, значення яких менше середнього геометричного значення.

4. Дано *n* дійсних чисел: $x_1, x_2, \ldots, x_n$. Знайти найбільше серед елементів масиву та адресу його місця пербування.
5. Дано *n* дійсних чисел: $x_1, x_2, \ldots, x_n$. Знайти найменше серед елементів масиву та адресу його місця пербування.
6. Дано *n* дійсних чисел: $x_1, x_2, \ldots, x_n$. Знайти найбільше серед від'ємних елементів масиву та адресу його місця пербування.
7. Дано *n* дійсних чисел: $x_1, x_2, \ldots, x_n$. Знайти найменше серед додатних елементів масиву та адресу його місця пербування.
8. Знайти добуток від'ємних елементів вектора $a \in R^n$. Вивести на екран всі від'ємні елементи та адреси їх перебування в масиві.
9. Знайти кількість від'ємних елементів у векторі $x \in R^n$, які розташовані після першого додатного. Вивести на екран адресу першого додатного та адреси від'ємних елементів, які розташовані після першого додатного.
10. Знайти найбільший елемент серед елементів вектора $x \in R^n$ з парними індексами.
11. У векторі $x \in R^n$ всі елементи, які більші за середнє арифметичне замінити нулями. Вивести адреси цих елементів.
12. Дано: $n \in N, x, y \in R^n$. Побудувати вектор z, який містить спочатку додатні координати вектора *x,* а потім додатні координати вектора *y*.
13. Дано одновимірний масив дійсних чисел. Знайти суму елементів, які розташовані до першого від'ємного елемента та вивести їх адреси.
14. Дано одновимірний масив дійсних чисел X. В цьому масиві поміняти місцями елементи, що розташовані симетрично відносно середини.

15. Перевірити чи впорядковані за зростанням елементи масиву цілих чисел до першого від‘ємного елементу. Вивести адресу першого від‘ємного елементу.

1.4. Зміст|вміст| звіту:

1. Прізвище та ім’я студента.
2. Номер і назва лабораторної роботи.
3. Мета роботи.
4. Номер індивідуального завдання. Текст завдання (постановка завдання).
5. Лістинг програми.
6. Скриншот робочої програми.
7. Висновок про засвоєнні знання та вміння.

1.5. Питання для самоперевірки

1. Що визначає вказівка типу даних?
2. Що визначає поняття «фізична структура даних»?
3. Що таке логічна структура даних?
4. Скільки рівнів опису структури даних існує? Охарактеризуйте їх.
5. За якими критеріями можна класифікувати структури даних?
6. Чим характеризуються статичні змінні?
7. Які змінні використовуються для зберження даних, які створюються і знищуються в процесі виконання програми ?
8. Як здійснюється доступ до комірок пам’яті?
9. З чого складається адреса комірки пам’яті?
10. Яку структуру має пам’ять комп’ютера, яку системи програмування виділяють для роботи програм?
11. Пам’ять під програмний код та дані є _____, а програмний стек та купа – _____ пам’яттю.
12. Що являє собою небезпечний (незахищений) код?
13. За допомогою якого ключового слова визначається небезпечний контекст виконання в C#-програмах?

14. Як можна встановити режим використання небезпечного коду шляхом настройки середовища Visual Studio?
15. Охарактеризуйте поняття вказівника.
16. Вкажіть особливості вказівників в C#-програмах.
17. Опишіть способи ініціалізації вказівників.
18. Перелічіть основні операції над вказівниками.
19. Опишіть призначення оператора fixed.
20. В чому полягає порівняння вказівників?
21. Як встановлюється порядок виконання операцій над вказівниками?
22. Опишіть призначення оператора stackalloc.

Лабораторна робота № 2. Динамічні структури даних. Однозв'язні списки. Визначення лінійних списків. Формування, доступ до елементів, виведення. Вставка, пошук, видалення елементів у однозв'язних списках.

Мета роботи: набути навичок створення та обробки однозв'язних лінійних списків.

Послідовність виконання роботи:

1. Ознайомитись із теоретичними відомостями. (*Актуалізація опорних знань*).
2. Виконати програмування програм за поданими прикладами. Результат представити викладачу (*Застосування набутих знань*).
3. Виконати варіант самостійної роботи. (*Закріплення набутих знань*).
4. Оформити звіт на виконану роботу. (*Узагальнення та систематизація набутих знань*).
5. Захист звітів, відповіді на запитання.

2.1. Теоретичні відомості

Динамічні структури даних. У мові C# є засоби створення динамічних структур даних, які дозволяють під час виконання програми створювати об'єкти, виділяти для них пам'ять, звільняти пам'ять, коли в ній зникає необхідність. Якщо до початку роботи з даними неможливо визначити, скільки пам'яті потрібно для їх зберігання, пам'ять слід розподіляти під час виконання програми в міру необхідності окремими блоками. Блоки зв'язуються один з одним за допомогою покажчиків. Такий спосіб організації даних називається динамічною структурою даних, оскільки вона розміщується в динамічній пам'яті і її розмір змінюється під час виконання програми.

Динамічні структури даних − це структури даних, пам'ять під які виділяється і звільняється в міру необхідності.

Динамічні структури даних в процесі існування в пам'яті можуть змінювати не тільки число складових їх елементів, а й характер зв'язків між елементами. При цьому не враховується зміна вмісту самих елементів даних. Така особливість динамічних структур, як мінливість їх розміру і характеру відносин між елементами призводить до того, що на етапі створення машинного коду програма-компілятор не може виділити для всієї структури в цілому ділянку пам'яті фіксованого розміру, а також не може порівнять з окремими компонентами структури конкретні адреси. Для вирішення проблеми адресації динамічних структур даних використовується метод, називаний динамічним розподілом пам'яті, тобто пам’ять під окремі елементи виділяється в момент, коли вони «починають існувати» в процесі виконання програми, а не під час компіляції. Компілятор в цьому випадку виділяє фіксований обсяг пам’яті для зберігання адреси елемента, який динамічно розміщується, а не самого елемента.

Динамічна структура даних характеризується тим, що:

- вона не має імені;
- їй виділяється пам'ять в процесі виконання програми;
- кількість елементів структури може не фіксуватися;
- розмірність структури може змінюватися в процесі виконання програми;
- в процесі виконання програми може змінюватися характер взаємозв'язку між елементами структури.

Кожній динамічній структурі даних відповідає статична змінна типу покажчик (її значення – адреса цього об'єкта), за допомогою якої здійснюється доступ до динамічної структури.

Самі динамічні величини не вимагають опису в програмі, оскільки під час компіляції пам'ять під них не виділяється. Під час компіляції пам’ять виділяється тільки під статичні змінні. Покажчики – це статичні змінні, тому вони вимагають опису.

Оскільки елементи динамічної структури розташовуються за непередбачуваними адресами пам'яті, адреса елемента такої структури не може бути обчислена з адреси початкового або попереднього елемента. Для встановлення зв’язку між елементами динамічної структури використовуються покажчики, через які встановлюються явні зв’язки між елементами. Таке уявлення даних в пам’яті називається пов’язаним.

Порядок роботи з динамічними структурами даних наступний:

1) створити (відвести місце в динамічній пам'яті);
2) працювати за допомогою покажчика;
3) видалити (звільнити зайняте структурою місце).

Класифікація динамічних структур даних. У багатьох задачах потрібно використовувати дані, в яких конфігурація, розміри і склад можуть змінюватися в процесі виконання програми. Для їх уявлення використовують динамічні інформаційні структури. До таких структур відносять:

– однозв’язний лінійний список;
– двозв’язний лінійний список;
– однозв’язний циклічний список;
– двозв’язний циклічний список;
– стек;
– черга.

Однозв’язний лінійний список – це список, в якому попередній компонент посилається на наступний.

Двозв’язний лінійний список – це список, в якому попередній компонент посилається на наступний, а наступний на попередній.

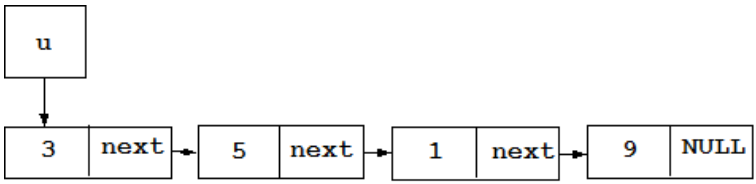
Однозв’язний циклічний список – це однозв’язний лінійний список, в якому останній компонент посилається на перший.

Двозв’язний циклічний список – це двозв’язний лінійний список, в якому останній компонент посилається на перший, а перший – на останній.

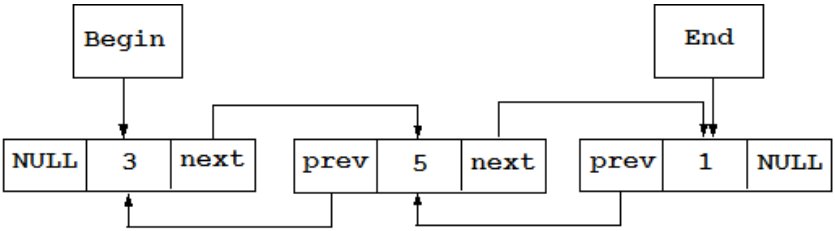
Стек – це однозв’язний лінійний список, в якому компоненти додаються та видаляються тільки з його вершини, тобто з початку списку.

Черга – це однозв’язний лінійний список, в якому компоненти додаються в кінець списку, а видаляються з початку списку.

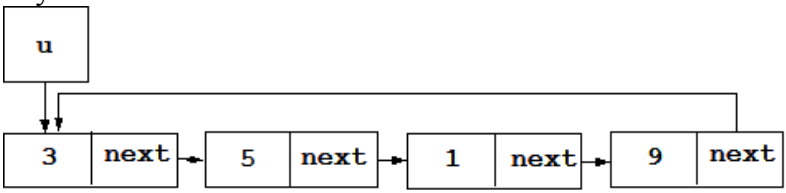
Графічне зображення однозв’язного лінійного списку:



Графічне зображення двозв’язного лінійного списку:



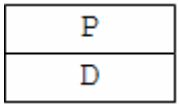
Графічне зображення циклічного однозв’язного лінійного списку



Як бачимо зі схем, компонент списку має специфічне представлення, а саме має поле для збереження інформації та поле покажчика на наступний елемент. Отже, компонента

зв’язного лінійного списку є структурою. Інформаційне поле може мати складну структуру і містити поля довільного типу, а покажчик повинен мати той самий тип, до якого належать компоненти списку. Покажчик в останньому компоненті лінійного списку має значення null – так позначається кінець списку.

Оголошення динамічних структур даних. Кожна компонента будь-якої динамічної структури є запис, що містить, принаймні, два поля: одне поле типу покажчик, а друге − для розміщення даних. У загальному випадку запис може містити не один, а кілька покажчиків і кілька полів даних. Поле даних може бути змінною, масивом або структурою. Для найкращого уявлення зобразимо окрему компоненту у вигляді:



де поле P – покажчик; поле D – дані.

Елемент динамічної структури складається з двох полів:

– інформаційного поля (поля даних), в якому містяться ті дані, заради яких і створюється структура; в загальному випадку інформаційне поле саме є інтегрованою структурою – вектором, масивом, іншою динамічною структурою і т.п.;
– адресного поля (поля зв'язок), в якому містяться один або кілька покажчиків, що зв'язує даний елемент з іншими елементами структури.

Оголошення елемента списку в C#:

для роботи з покажчиками в небезпечному коді:

```
struct List
{
   public int inf;
   unsafe public List* next;
}
```

для роботи з динамічними стуктурами даних під керівництвом CLR:

```
class List
{
    public int inf;
    public List next;
}
```

Доступ до даних в динамічних структурах. Елемент динамічної структури в кожен момент може або існувати, або бути відсутнім в пам'яті, тому його називають динамічним. Оскільки елементами динамічної структури є динамічні змінні, то єдиним засобом доступу до динамічних структур та їх елементів є покажчик (адреса) на місце їх поточного розташування в пам'яті. Таким чином, доступ до динамічних даних виконується спеціальним чином за допомогою покажчиків.

В C# вказівник можна утворити лише на типи за значенням. Для структур існує обмеження: структура не повинна містити типів за посиланням.

Для звернення до динамічної структури достатньо зберігати в пам'яті адресу першого елемента структури. Оскільки кожен елемент динамічної структури зберігає адресу наступного за ним елемента, можна, рухаючись від початкового елемента за адресами, отримати доступ до будь-якого елемента даної структури.

Доступ до даних в динамічних структурах здійснюється за допомогою операції «стрілка» (->), яку називають операцією непрямого вибору елемента структурного об’єкта, що адресується покажчиком. Вона забезпечує доступ до елемента структури через адресу її покажчика того ж структурного типу.

Формат застосування даної операції наступний:

ПокажчикНаСтруктуру-> Ім’яЕлемента;

Операція «стрілка» (->) двомісна. Застосовується для доступу до елемента, що задається правим операндом тієї структури, яку адресує лівий операнд. В якості лівого

операнда повинен бути покажчик на структуру, а в якості правого – ім’я елемента цієї структури. Наприклад:

```
P->Data;
p-Next;
```

Однозв’язні списки. ***Списком*** називається впорядкована множина, що складається зі змінного числа елементів, до яких застосовані операції включення, виключення. Список, що відображає відносини сусідства між елементами, називається лінійним. ***Довжина списку*** дорівнює числу елементів, що містяться у списку, список нульової довжини називається порожнім списком. Списки є спосіб організації структури даних, за якої елементи деякого типу утворюють ланцюжок. Для зв’язування елементів у списку використовують систему покажчиків. У мінімальному випадку будь-який елемент лінійного списку має один покажчик, який вказує на наступний елемент у списку або є порожнім покажчиком, що інтерпретується як кінець списку.

Структура, елементами якої служать записи з одним і тим самим форматом, пов’язані один з одним за допомогою покажчиків, що зберігаються у самих елементах, називають ***зв’язним списком***. У зв’язному списку елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а покажчиками, що входять до складу елементів списку. Кожен список має особливий елемент, називаний покажчиком початку списку (головою списку), який зазвичай за змістом відмінний від інших елементів. В полі покажчика останнього елемента списку знаходиться спеціальна позначка NULL, яка свідчить про кінець списку.

Лінійні зв’язні списки є найпростішими динамічними структурами даних. З усього різноманіття зв'язних списків можна виділити основні:

– односпрямовані (однозв'язні) списки;
– двоспрямовані (двозв’язні) списки;
– циклічні (кільцеві) списки.

В основному вони відрізняються видом взаємозв’язку

елементів і/або допустимими операціями.

Найбільш простою динамічною структурою є односпрямований список, елементами якого служать об'єкти структурного типу.

Односпрямований (однозв'язний) список − це структура даних, що являє собою послідовність елементів, в кожному з яких зберігається значення і покажчик на наступний елемент списку (рис. 2.1). В останньому елементі покажчик на наступний елемент дорівнює NULL.

Опис найпростішого елемента такого списку виглядає наступним чином:

struct ім'я_типу {інформаційне поле; адресне поле; };

де інформаційне поле − це поле будь-якого, раніше оголошеного або стандартного типу;

адресне поле − це покажчик на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента списку.

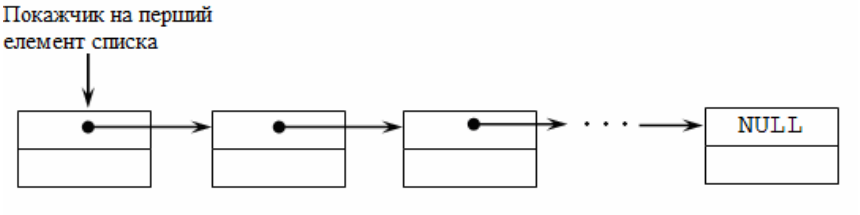


Рис. 2.1. Лінійні односпрямованими списки

Наприклад:

для роботи з покажчиками в небезпечному коді:

```
struct Node
{
    public int inf;
    unsafe public Node* next;
}
```

Інформаційних полів може бути кілька. Наприклад:

```
struct P
{
    public unsafe P* next;//адресне поле
    public char name;//інформаційне поле
```

```
        public int age;//інформаційне поле
    };
```

Кожен елемент списку містить ключ, який ідентифікує цей елемент. Ключ зазвичай буває або цілим числом, або рядком. Основними операціями, які здійснюються з односпрямованим списками, є:

– створення списку;
– друк (перегляд) списку;
– включення елемента у список;
– виключеня елемента зі списку;
– пошук елемента у списку;
– перевірка відсутності елементів у списку (NULL);
– видалення списку.

Особливу увагу слід звернути на те, що при виконанні будь-яких операцій з лінійним односпрямованим списком необхідно забезпечувати позиціонування будь-якого покажчика на перший елемент. В іншому випадку частина або весь список буде недоступний.

Розглянемо докладніше кожну з наведених операцій. Для опису алгоритмів цих основних операцій використовуватимемо оголошення:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp2
{
    class Program
    {
        public  struct Single_List   //структура даних
        {
            public int Data; //інформаційне поле
            public unsafe Single_List* Next; //адресне поле
        };
```

```
        static unsafe void Main(string[] args)
        {
           //покажчик на перший елемент списку
            Single_List* Head=null;
           //покажчик на поточний элемент списку (за
необхідносттю)
            Single_List* Current=null;
        }
    }
}
```

Формування, доступ до елементів, виведення.

Створення односпрямованого списку. Розглянемо приклад створення лінійного однозв'язного списку з послідовності цілих чисел, яка закінчується нулем (за ознаку закінчення послідовності можна прийняти довільне число).

Для того, щоб створити список, потрібно створити спочатку перший елемент списку, а потім додавати нові елементи в хвіст списку:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp2
{
    class Program
    {
        public struct Single_List   //структура даних
        {
            public int Data; //інформаційне поле
            public unsafe Single_List* Next; //адресне поле
        };

        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Single_List* Head = null;
            //покажчик на останній элемент списку (за
необхідностю)
            Single_List* Last = null;
            //Введення першого елемента списку
```

```
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;  //Лічильник елементів списку
            while (n!=0) //Поки не введено признака кінця
послідовності
            {
                //Створюємо елемент списку
                Single_List*  Current = stackalloc
Single_List[sizeof(Single_List)];
                Current->Data = n; //Заносимо значення в
інформаційне поле
                Current->Next = null; //В адресне поле заносимо
0

                if (Head == null)  //якщо елемент перший,
                {                  //він становиться головою та
хвостом списку
                    Head = Last= Current;
                }
                else
                {
                    //Додавання елемента в кінець списку
                    Last->Next = Current;
                    Current->Prev = Last;
                    Last = Current;
                }                //Введення нового елемента
                Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
                n = int.Parse(Console.ReadLine());
                count++;
            }
            Console.WriteLine("Кількість елементів в стеку:
{0}", count);
            Console.ReadLine();
        }
    }
}
```

Друк (перегляд) односпрямованого списку

Операція друку списку полягає в послідовному перегляді всіх елементів списку і виведення їх значень на екран. Для обробки списку створюються цикл, в якому потрібно

переставляти вказівник на наступний елемент списку до тих пір, поки покажчик не стане дорівнювати *null*, тобто буде досягнутий кінець списку.

```
//Виведення елементів списку
            Single_List*  Cur = Head;
            while (Cur != null) //Поки не кінець списку
            {
                Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
                Cur = Cur->Next; //Перехід до наступного
елемента
            }
```

Включення, пошук, видалення елементів у однозв'язних списках.

Включення елемента до односпрямованого списку. У динамічні структури легко додавати елементи, тому що для цього достатньо змінити значення адресних полів. Алгоритми включення першого, останнього і внутрішніх елементів списку різні. Тому у функції, що реалізує дану операцію, спочатку здійснюється перевірка, на яке місце вставляється елемент. Далі реалізується відповідний алгоритм додавання (рис. 2.2).

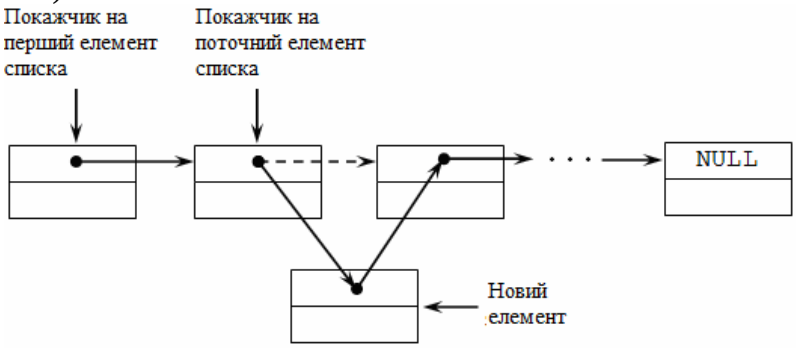


Рис.2.2. Включення елемента в односпрямований список

```
//включення елемента з заданим номером до односпрямованого
списку
            Console.WriteLine("ВВедіть місце елемента: ");
            int Place = int.Parse(Console.ReadLine());
```

```
            Console.WriteLine("ВВедіть елемент: ");
            n = int.Parse(Console.ReadLine());
            //Створюємо новий елемент списку
            Single_List*  NewItem = stackalloc
Single_List[sizeof(Single_List)];
            NewItem->Data = n; //Заносимо значення в
інформаційне поле
            NewItem->Next = null; //В адресне поле заносимо 0
            if (Head == null)  //якщо список порожній,
            {                  //елемент становиться головою та
хвостом списку
                Head = Last = NewItem;
            }
            else //список не порожній
            {
                if (Place == 0) //Якщо потрібно вставити елемент
на перше місце
                {               //новий елемент становиться
головою списка
                    NewItem->Next = Head; //В адресне поле
нового елемента записуємо адресу
                                          //попереднього першого
елемента
                    Head = NewItem; //Новий елемент становиться
першим елементом списку
                }
                else
                {
                    Single_List* Curr = Head;
                    //Цикл для визначення місця вставлення
нового елемента
                    for (int i = 1; i < Place && Curr->Next !=
null; i++)
                    {
                        Curr = Curr->Next;
                    }

                    if (Curr != null) //Вставлення всередину
списка
                    {
                        NewItem->Next = Curr->Next;
                        Curr->Next = NewItem;
                    }
                    else     //Вставлення в кінець списка
```

```
                {
                    Curr->Next = NewItem;
                }
            }
        }
```

Виключеня елемента з односпрямованого списку. З динамічних структур можна виключати елементи, тому що для цього достатньо змінити значення адресних полів. Операція виключення елемента односпрямованого списку здійснює виключення елемента, на який встановлено покажчик поточного елемента. Після виключення покажчик поточного елемента встановлюється на попередній елемент списку або на новий початок списку, якщо видаляється перший.

Алгоритми виключення першого і наступних елементів списку відрізняються один від одного. Тому у функції, що реалізує дану операцію, здійснюється перевірка, який об'єкт був видалений. Далі реалізується відповідний алгоритм виключення (рис. 2.3).

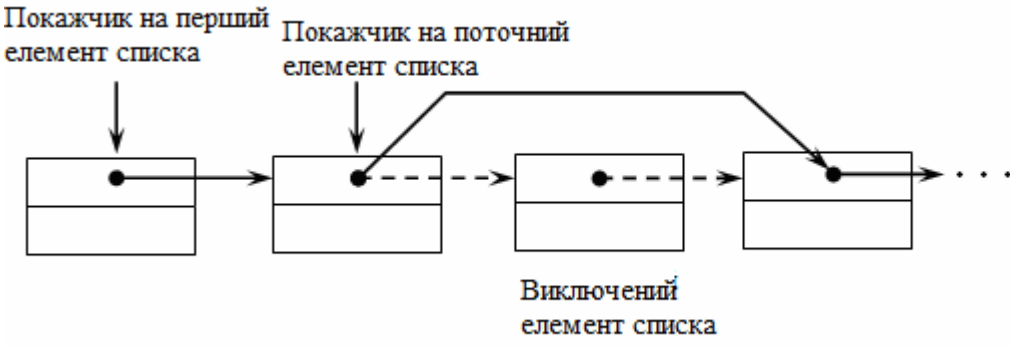


Рис. 2.3. Виключення елемента з односпрямованого списку

```
//виключення елемента з заданим номером з односпрямованого
списку
            Console.WriteLine("ВВедіть номер  елемента для
      виключення: ");
            int Place = int.Parse(Console.ReadLine());
            Single_List* Curr1 = Head;
            //Цикл для визначення місця видалення елемента
```

```
            for (int i = 1; i < Place && Curr1->Next != null;
i++)
            {
                Curr1 = Curr1->Next;
            }
            if (Curr1 != null) //перевірка на коректність
            {
                if (Curr1 == Head)
                { //виключення першого элемента
                    Head = Head->Next;
                }
                else
                { //виключення не першого елемента
                    Single_List* ptr = Head;
                    while (ptr->Next != Curr1) ptr = ptr->Next;
                    ptr->Next = Curr1->Next;
                    Curr1 = ptr;
                }
```

2.2. Приклади виконання завдань.

Завдання 2.2.1. Сворити однозв'язний лінійний список з послідовності цілих чисел, яка закінчується нулем (за ознаку закінчення послідовності можна прийняти довільне число). Вставити елемент списку на задане місце. Видалити елемент з певним номером.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp2
{
    class Program
    {
        public struct Single_List   //структура даних
        {
            public int Data; //інформаційне поле
            public unsafe Single_List* Next; //адресне поле
        };
```

```
        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Single_List* Head = null;
            //покажчик на останній элемент списку (за
необхідностю)
            Single_List* Last = null;
            //Створення односпрямованого списку
            //Введення першого елемента списку
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;  //Лічильник елементів списку
            while (n != 0) //Поки не введено признака кінця
послідовності
            {
                //Створюємо новий елемент списку
                Single_List* Current = stackalloc
Single_List[sizeof(Single_List)];
                Current->Data = n; //Заносимо значення в
інформаційне поле
                Current->Next = null; //В адресне поле заносимо
0

                if (Head == null)  //якщо елемент перший,
                {                  //він становиться головою та
хвостом списку
                    Head = Last = Current;
                }
                else
                {
                    //Додавання елемента в кінець списку
                    Last->Next = Current;
                    Current->Prev = Last;
                    Last = Current;
                }                //Введення нового елемента
                Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
                n = int.Parse(Console.ReadLine());
                count++;
            }
            Console.WriteLine("Кількість елементів в списку:
{0}", count);
```

```
            //включення елемента з заданим номером до
односпрямованого списку
            Console.WriteLine("ВВедіть місце елемента: ");
            int Place = int.Parse(Console.ReadLine());
            Console.WriteLine("ВВедіть елемент: ");
            n = int.Parse(Console.ReadLine());
            //Створюємо новий елемент списку
            Single_List* NewItem = stackalloc
Single_List[sizeof(Single_List)];
            NewItem->Data = n; //Заносимо значення в
інформаційне поле
            NewItem->Next = null; //В адресне поле заносимо 0
            if (Head == null)  //якщо список порожній,
            {                  //елемент становиться головою та
хвостом списку
                Head = Last = NewItem;
            }
            else //список не порожній
            {
                if (Place == 0) //Якщо потрібно вставити елемент
на перше місце
                {               //новий елемент становиться
головою списка
                    NewItem->Next = Head; //В адресне поле
нового елемента записуємо адресу попередньої голови
                    Head = NewItem; //Новий елемент становиться
першим елементом списку
                }
                else
                {
                    Single_List* Curr = Head;
                    //Цикл для визначення місця вставлення
нового елемента
                    for (int i = 1; i < Place && Curr->Next !=
null; i++)
                    {
                        Console.WriteLine(Curr->Data);
                        Curr = Curr->Next;
                    }

                    if (Curr != null) //Вставлення в середину
списка
                    {
```

```
                    Console.WriteLine(11);
                    NewItem->Next = Curr->Next;
                    Curr->Next = NewItem;
                }
                else     //Вставлення в кінець списка
                {
                    Console.WriteLine(22);
                    //Curr = Curr->Next;
                    Curr->Next = NewItem;
                }
            }
        }
        //Виведення елементів списку
        Single_List* Cur = Head;
        while (Cur != null) //Поки не кінець списку
        {
            Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
            Cur = Cur->Next; //Перехід до наступного
елемента
        }
        //виключення елемента з заданим номером з
односпрямованого списку
        Console.WriteLine("ВВедіть номер  елемента для
виключення: ");
        Place = int.Parse(Console.ReadLine());
        Single_List* Curr1 = Head;
        //Цикл для визначення місця видалення елемента
        for (int i = 1; i < Place && Curr1->Next != null;
i++)
        {
            Curr1 = Curr1->Next;
        }
        if (Curr1 != null)
        {//перевірка на коректність
            if (Curr1 == Head)
            {//виключення першого элемента
                Head = Head->Next;
            }
            else
            {//виключення не першого елемента
                Single_List* ptr = Head;
                while (ptr->Next != Curr1) ptr = ptr->Next;
                ptr->Next = Curr1->Next;
```

```
                    Curr1 = ptr;
                }
                //Виведення елементів списку
                Cur = Head;
                while (Cur != null) //Поки не кінець списку
                {
                    Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
                    Cur = Cur->Next; //Перехід до наступного
елемента
                }
                Console.ReadLine();
            }
        }
    }
}
```

2.3. Індивідуальні завдання

Потрібно реалізувати кожне завдання у відповідності з наведеними етапами:

1. вивчити словесну постановку задачі, виділяючи при цьому всі види даних;
2. сформулювати математичну постановку задачі;
3. обрати метод розв'язання задачі, якщо це необхідно;
4. розробити графічну схему алгоритму;
5. записати розроблений алгоритм мовою програмування;
6. розробити контрольний тест до програми;
7. налагодити програму;
8. представити звіт до роботи.

Варіант кожного завдання обирається за номером студента в журналі.

1 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням 55.

2 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку елемент після кожного елемента з від'ємним значенням.

3 Створити лінійний односпрямований список з дійсних чисел.

Вставити в список число 1.5 після кожного елемента з від'ємним значенням.

4 Створити лінійний односпрямований список з дійсних чисел. Визначити суму елементів списку зі значеннями більше або рівними 15.

5 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку перший елемент менший за модулем 5.

6 Створити лінійний односпрямований список з дійсних чисел. Продублювати в списку перший позитивний елемент (якщо такого немає, залишити список без зміни).

7 Створити лінійний односпрямований список з цілих чисел. Видалити зі списку перший парний елемент, що стоїть на непарній позиції.

8 Створити лінійний односпрямований список з цілих чисел. Вставити в список число 66 після кожного елемента з від'ємним значенням.

9 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням -2. Вставити число 33 в кінець списку.

10 Створити лінійний односпрямований список з цілих чисел. Видалити зі списку елемент після кожного елемента, рівного 4. Вставити число 0 перед кожним числом 1.

11 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку всі числа від 2 до 5.Вставити в список число 1.5 після кожного елемента з від'ємним значенням.

12 Створити лінійний односпрямований список з цілих чисел. Визначити суму елементів списку зі значеннями більше або рівними 15. Видалити зі списку всі значення, які менше 5.

13 Створити лінійний односпрямований список з символів. Видалити зі списку перший елемент, код якого менше 48. Вставити символ % після кожної цифри.

14 Створити лінійний односпрямований список з дійсних чисел. Вставити перший позитивний елемент списку після кожного негативного числа (якщо такого немає, залишити список без зміни).

15 Створити лінійний односпрямований список з цілих чисел. Вставити в список останній парний елемент після кожного непарного елемента.
16 Створити лінійний двонаправлений список з цілих чисел. Вставити в список число 11 після кожного елемента, рівного 9.
17 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням в інтервалі від 10 до 20.
18 Створити лінійний односпрямований список із символів. Видалити зі списку елемент після кожного символу &.
19 Створити лінійний односпрямований список з дійсних чисел. Вставити в список число 13.5 після першого елемента зі значенням більшим 2.
20 Створити лінійний односпрямований список з дійсних чисел. Визначити середнє значення елементів списку зі значеннями менше або рівними 15. Видалити зі списку елементи, які більше 25.
21 Створити лінійний односпрямований список з цілих чисел. Видалити зі списку перший елемент, більший числа 4. Вставити в список число 10 перед кожним числом, рівним 15.
22 Створити лінійний односпрямований список з дійсних чисел. Вставити в список перший негативний елемент перед кожним числом, рівним 20 (якщо таких немає, залишити список без зміни).
23 Створити лінійний односпрямований список з цілих чисел. Видалити зі списку кожен елемент, кратний трьом. Вставити в список число 88 після кожної пари рівних поруч стоять чисел.
24 Створити лінійний односпрямований список з цілих чисел. Вставити в список число 25 перед кожним елементом з позитивним значенням. Видалити зі списку всі негативні числа.
25 Створити лінійний односпрямований список з символів. Видалити зі списку елемент перед кожним символом ^.

Визначити кількість символів * в списку.

26 Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку елементи, у яких дробова частина більше 0,5.

27 Створити лінійний односпрямований список з цілих чисел. Вставити в список число 0 перед кожним числом від 2 до 7. Визначити суму чисел більших за 7.

28 Створити лінійний односпрямований список з дійсних чисел. Визначити максимальне з елементів списку і вставити його після кожного елемента зі значенням 1.

29 Створити лінійний односпрямований список з цілих чисел. Видалити зі списку перший елемент кратний 5. Вставити число 44 перед кожним числом кратним 7.

30 Створити лінійний односпрямований список з дійсних чисел. Обчислити середнє значення елементів списку і вставити число 11 перед кожним числом, більшим за середнє значення.

2.4. Зміст|вміст| звіту:

1. Прізвище та ім’я студента.
2. Номер і назва лабораторної роботи.
3. Мета роботи.
4. Номер індивідуального завдання. Текст завдання (постановка завдання).
5. Лістинг програми.
6. Скриншот робочої програми.
7. Висновок про засвоєнні знання та вміння.

2.5. Питання для самоперевірки

1. Чому в програмах розмір пам’яті під статичні змінні повинен бути визначений на етапі компіляції?
2. За рахунок яких ресурсів виділяється пам’ять під динамічні структури?
3. Чому динамічні структури не вимагають власного опису в програмі?

4. Як розташовуються в пам’яті динамічні величини?
5. Як здійснюється доступ до динамічних структур з програмного коду?
6. Як зв'язуються між собою елементи динамічної структури?
7. У чому основна відмінність суміжного і зв’язного подання даних?
8. Якого типу може бути поле даних в динамічній структурі?
9. Чому для звернення до динамічної структури досить зберігати в пам'яті адресу її першого елемента?
10. За рахунок чого робота з динамічними даними уповільнює виконання програми?
11. Чи кожен список є зв’язним? Обгрунтуйте відповідь.
12. У чому відмінність першого елемента односпрямованого списку від інших елементів цього ж списку?
13. У чому відмінність останнього елемента односпрямованого списку від інших елементів цього ж списку?
14. Чому при роботі з односпрямованим списком необхідне позиціонування на перший елемент списку?
15. У чому принципові відмінності виконання додавання (видалення) елемента на першу і будь-яку іншу позиції в односпрямованому списку?
16. З якою метою в програмах виконується перевірка на порожнину односпрямованого (двоспрямованого) списку?

Лабораторна робота № 3. Двозв'язні лінійні списки. Кільця. Програмна реалізація двонаправлених списків і кілець. Створення, доступ до елементів, відображення. Вставлення, пошук, видалення елементів у двозв'язних списках та кільцях.

Мета роботи: набути навичок створення та обробки двозв'язних лінійних списків та кілець.

Послідовність виконання роботи:

1. Ознайомитись із теоретичними відомостями. (*Актуалізація опорних знань*).
2. Виконати програмування програм за поданими прикладами. Результат представити викладачу (*Застосування набутих знань*).
3. Виконати варіант самостійної роботи. (*Закріплення набутих знань*).
4. Оформити звіт на виконану роботу. (*Узагальнення та систематизація набутих знань*).
5. Захист звітів, відповіді на запитання.

3.1. Теоретичні відомості

Двоспрямовані (двозв'язані) списки. Для прискорення багатьох операцій доцільно застосовувати переходи між елементами списку в обох напрямках. Це реалізується за допомогою двоспрямованих списків, які є складною динамічною структурою.

Двоспрямовані (двозв'язані) списки − це структура даних, що складається з послідовності елементів, кожен з яких містить інформаційну частину і два покажчика на сусідні елементи (рис. 3.1). При цьому два сусідні елементи повинні містити взаємні посилання один на одного.

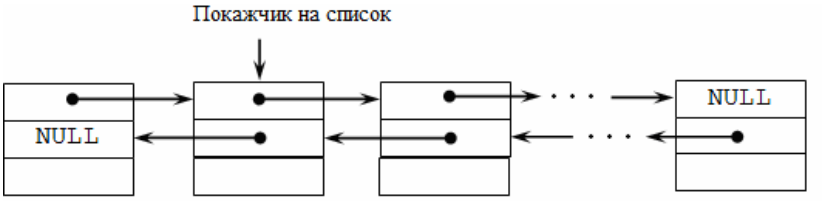


Рис.3.1. Двоспрямований (двозв’язаний) список

Опис найпростішого елемента такого списку виглядає наступним чином:

```
public unsafe struct ім’я_типу
{
    інформаційне поле;
    адресне поле1;
    адресне поле2;
};
```

де інформаційне поле – це поле будь-якого, раніше оголошеного або стандартного, типу;
адресне поле 1 – це покажчик на об’єкт того самого типу, що і визначена структура, в нього записується адреса наступного елемента списку;
адресне поле 2 – це покажчик на об'єкт того самого типу, що і визначена структура, в нього записується адреса попереднього елемента списку.

Наприклад:

```
//структура даних для організації роботи з двозв'язним списком
        public unsafe struct Single_List
        {
            public type Data; //інформаційне поле
            public Single_List* Next; //адресне поле для зв'язку
с наступним елементом
            public Single_List* Prev; //адресне поле для зв'язку
с попереднімелементом
        };
//покажчик на перший елемент списку
            Single_List* Head = null;
```

де *type* – тип інформаційного поля елемента списку;
**Next, *Prev* – покажчики на наступний і попередній елементи цієї структури відповідно. Змінна-покажчик *Head* задає список як єдиний програмний об'єкт, її значення – покажчик на перший (або титульний) елемент списку.

Основні операції, що виконуються над двоспрямованим списком, ті самі, що й для односпрямованого списку. Тому що двоспрямований список більш гнучкий, ніж односпрямований, то при включенні елемента у список, потрібно використовувати покажчик як на елемент, за яким відбувається включення, так і покажчик на елемент, перед яким відбувається включення. При виключення елемента зі списку потрібно використовувати як покажчик на сам виключається елемент, так і покажчики на попередній або наступний за виключається елементи. Але тому що елемент двоспрямованого списку має два покажчика, то при виконанні операцій включення/виключення елемента треба змінювати більше зв'язків, ніж в односпрямованому списку. Розглянемо основні операції, здійснювані з двоспрямованими списками, такі як:

- створення списку;
- друк (перегляд) списку;
- вставка елемента у список;
- видалення елемента зі списку;
- пошук елемента у списку;
- перевірка порожнього списку;
- видалення списку.

Особливу увагу слід звернути на те, що на відміну від односпрямованого списку тут немає необхідності забезпечувати позиціонування будь-якого покажчика саме на перший елемент списку, тому що завдяки двом вказівникам в елементах можна отримати доступ до будь-якого елементу списку з будь- якого іншого елемента, здійснюючи переходи в

прямому або зворотному напрямку. Однак за правилами хорошого тону програмування покажчик бажано ставити на заголовок списку.

Для опису алгоритмів цих основних операцій використовується наступне оголошення:

```
public unsafe struct Single_List
        {
            public int Data; //інформаційне поле
            public Single_List* Next; //адресне поле для зв'язку
с наступним елементом
            public Single_List* Prev; //адресне поле для зв'язку
с попереднімелементом
        };
        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Single_List* Head = null;
            //покажчик на поточний элемент списку (за
необхідностю)
            Single_List* Last = null;
        }
```

Створення двоспрямованого списку. Для того, щоб створити список, потрібно створити спочатку перший елемент списку, а потім додати до нього інші елементи. Додавання може виконуватися як на початок, так і на кінець списку.

Алгоритм додавання нового елемента в кінець списку аналогічний до такої операції для односпрямованого списку, але потрібно пам’ятати, що елемент двоспрямованого списку має два адресних поля (в поле зв’язку нового елемента з наступним елементом потрібно занести null, а в поле зв’язку з попередним елементом занести адресу останнього елемента списку).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace ConsoleApp2
{
    class Program
    {
        //структура даних для організації роботи з двозв'язним
списком
        public unsafe struct Double_List
        {
            public int Data; //інформаційне поле
            public Double_List* Next; //адресне поле для зв'язку
с наступним елементом
            public Double_List* Prev; //адресне поле для зв'язку
с попереднімелементом
        };
        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Double_List* Head = null;
            //покажчик на останній элемент списку (за
необхідностю)
            Double_List* Last = null;
            //Введення першого елемента списку
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;  //Лічильник елементів списку
            while (n != 0) //Поки не введено признака кінця
послідовності
            {
                //Створюємо новий елемент списку
                Double_List* Current = stackalloc
Double_List[sizeof(Double_List)];
                Current->Data = n; //Заносимо значення в
інформаційне поле
                Current->Next = null; //В адресне поле заносимо
0
                Current->Prev = null;

                if (Head == null)  //якщо елемент перший,
                {                  //він становиться головою та
хвостом списку
                    Head = Last = Current;

                }
```

```
                else
                {
                    //Додавання елемента в кінець списку
                    Last->Next = Current;
                    Current->Prev = Last;
                    Last = Current;
                }
                //Введення нового елемента
                Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
                n = int.Parse(Console.ReadLine());
                count++;
            }
            Console.WriteLine("Кількість елементів в списку:
{0}", count);
            Console.WriteLine("Виведення елементів  з початку
списку");
                Console.ReadLine();

        }
    }
}
```

Алгоритм **додавання нового елемента на початок списку** такий:

1. Перевірити чи список не порожній.
2. Якщо список порожній, то створити перший елемент списку, який буде головою і хвостом списку та занести значення в інформаційне поле.
3. Якщо у списку вже є елементи, то зробити такі дії:
   3.1. Створити новий елемент.
   3.2. Занести значення в інформаційне поле.
   3.3. В поле зв’язку з наступним елементом нового елемента занести адресу першого елемента списку.
   3.4. В поле зв’язку з попереднім елементом першого елемента списку занести адресу нового елемента.
   3.5. Зробити першим елементом списку новий елемент.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp2
{
    class Program
    {
        //структура даних для організації роботи з двозв'язним
списком
        public unsafe struct Double_List
        {
            public int Data; //інформаційне поле
            public Double_List* Next; //адресне поле для зв'язку
с наступним елементом
            public Double_List* Prev; //адресне поле для зв'язку
с попереднімелементом
        };
        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Double_List* Head = null;
            //покажчик на останній элемент списку (за
необхідностю)
            Double_List* Last = null;
            //Введення першого елемента списку
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;  //Лічильник елементів списку
            while (n != 0) //Поки не введено признака кінця
послідовності
            {
                //Створюємо новий елемент списку
                Double_List* Current = stackalloc
Double_List[sizeof(Double_List)];
                Current->Data = n; //Заносимо значення в
інформаційне поле
                Current->Next = null; //В адресне поле заносимо
0
                Current->Prev = null;

                if (Head == null)  //якщо елемент перший,
                {                  //він становиться головою та
хвостом списку
```

```
                    Head = Last = Current;

                }
                else
                {
                    //Додавання елемента на початок списку
                    Current->Next = Head;
                    Head->Prev = Current;
                    Head = Current;
                }
                //Введення нового елемента
                Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
                n = int.Parse(Console.ReadLine());
                count++;
            }
            Console.WriteLine("Кількість елементів в списку:
{0}", count);
            Console.WriteLine("Виведення елементів  з початку
списку");
            Console.ReadLine();

        }
    }
}
```

Друк (перегляд) двоспрямованого списку. Операція друку списку для двоспрямованого списку реалізується абсолютно аналогічно відповідної функції для односпрямованого списку. Переглядати двоспрямований список можна в обох напрямках.

```
//Виведення елементів  з початку списку
Double_List* Cur = Head; //Встановлюємо поточний вказівник на
початок списку
while (Cur != null) //Поки не кінець списку
{
    Console.WriteLine(Cur->Data); //Виведення значення
інформаційного поля
    Cur = Cur->Next; //Перехід до наступного елемента
}

Console.WriteLine("Виведення елементів  з кінця списку");
```

```
//Виведення елементів  з кінця списку
Cur = Last; //Встановлюємо поточний вказівник на кінець списку
while (Cur != null) //Поки не кінець списку
{
    Console.WriteLine(Cur->Data); //Виведення значення
інформаційного поля
    Cur = Cur->Prev; //Перехід до наступного елемента
}
```

Включення елемента в двоспрямований список. У динамічних структур легко додавати елементи, тому що для цього достатньо змінити значення адресних полів. Операція включення реалізовується аналогічно функції включення для односпрямованого списку, тільки з урахуванням особливостей двоспрямованого списку (рис. 3.2).

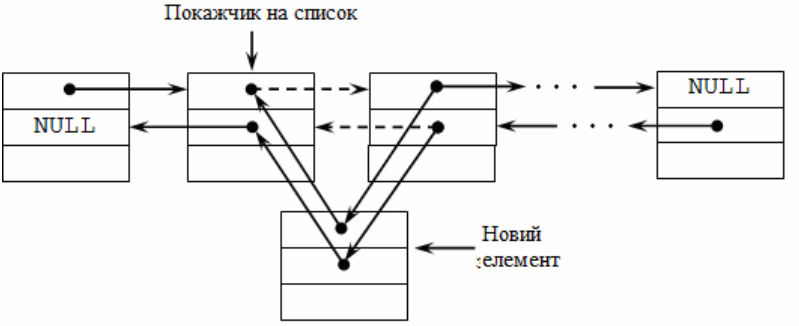


Рис. 3.2. Включення елемента в двоспрямований список

```
   //включення елемента з заданим номером до односпрямованого
списку
   Console.WriteLine("ВВедіть місце елемента: ");
   int Place = int.Parse(Console.ReadLine());
   Console.WriteLine("ВВедіть елемент: ");
   n = int.Parse(Console.ReadLine());
   //Створюємо новий елемент списку
   Double_List* NewItem = stackalloc
Double_List[sizeof(Double_List)];
   NewItem->Data = n; //Заносимо значення в інформаційне поле
   NewItem->Next = null; //В адресне поле заносимо 0
   NewItem->Prev = null;
   if (Head == null)  //якщо список порожній,
   {      //елемент становиться головою та хвостом списку
```

```
        Head = Last = NewItem;
    }
    else //список не порожній
    {
        if (Place == 0) //Якщо потрібно вставити елемент на перше
місце
        {   //новий елемент становиться головою списка
            NewItem->Next = Head; //В адресне поле нового
елемента записуємо адресу попередньої голови
            Head->Prev = NewItem;
            Head = NewItem; //Новий елемент становиться першим
елементом списку
        }
        else
        {
            Double_List* Curr = Head;
            //Цикл для визначення місця вставлення нового
елемента
            for (int i = 1; i < Place && Curr->Next != null; i++)
            {
                Curr = Curr->Next;
            }
            if (Curr->Next != null) //Вставлення в середину
списка
            {
                Curr->Next->Prev = NewItem;
                NewItem->Next = Curr->Next;
                Curr->Next = NewItem;
                NewItem->Prev = Curr;
            }
            else     //Вставлення в кінець списка
            {
                //Curr->Next = NewItem;
                Last->Next = NewItem;
                NewItem->Prev = Last;
                Last = NewItem;
            }
        }
    }
```

Виключення елемента з двоспрямованого списку. З динамічних структур можна виключати елементи, тому що для цього достатньо змінити значення адресних полів. Операція виключення елемента з двоспрямованого списку

здійснюється багато в чому аналогічно виключення з односпрямованого списку (рис. 2.6).

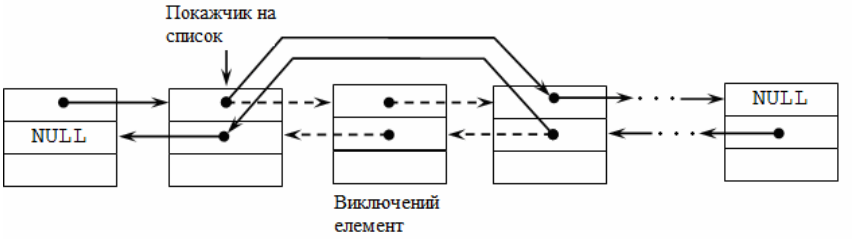


Рисунок 2.6 − Виключення елемента з двоспрямованого списку

Під час виключення елемента з двоспрямованого списку потрібно розглянути три випадка:

1. виключення першого елемента;
2. виключаємо останній елемент;
3. виключення внутрішнього елемента списку.

Виключення першого елемента. Встановлюємо поточний вказівник на перший елемент списку:

```
Current = Head;
```

та перевіряємо чи список не порожній:

```
if (Current != null)
```

Якщо умова виконується, потрібно зробити такі дії:

1. першим елементом зробити елемент, адреса якого зберігається в полі наступного елемента голови списку:

```
Head = Head->Next;
```

2. в поле попереднього елемента нової голови списку записати значення *null*:

```
Head->Prev=null;
```

Розглянемо випадок **виключення останнього елемента** двоспрямованого списку.

Встановлюємо поточний вказівник на перший елемент списку:

```
Current = Head;
```

та перевіряємо чи список не порожній:

```
if (Current != null)
```

Знаходимо останній елемент:

```
while (Current->Next != null) //Поки не кінець списку
{
        Current = Current->Next; //Перехід до наступного
    елемента
}
```

та видаляємо його

```
        Current->Prev->Next= null;
```

Виключення не першого та не останнього елемента. Для виключення внутрішнього елемента списку потрібно зробити такі дії:

1. в поле Next (`Curr1->Prev->Next`) попереднього елемента записуємо адресу наступного за елементом, що виключається:

```
Curr1->Prev->Next = Curr1->Next;
```

2. в поле Prev (`Curr1->Next->Prev`) наступного елемента записуємо адресу попереднього елемента, за яким слідує елемент, що виключається:

```
Curr1->Next->Prev = Curr1->Prev;
```

3.2. Приклади виконання завдань.

Приклад 3.1.1. Написати програму для створення двоспрямованого списку, елементи якого додаються з голови списку. Вивести елементи на екран. Додати та видалити елемент списку в задану позицію. Результати вивести на екран.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp2
{
    class Program
    {
        //структура даних для організації роботи з двозв'язним
списком
        public unsafe struct Double_List
```

```
        {
            public int Data; //інформаційне поле
            public Double_List* Next; //адресне поле для зв'язку
с наступним елементом
            public Double_List* Prev; //адресне поле для зв'язку
с попереднімелементом
        };
        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Double_List* Head = null;
            //покажчик на останній элемент списку (за
необхідностю)
            Double_List* Last = null;
            //Введення першого елемента списку
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;  //Лічильник елементів списку
            while (n != 0) //Поки не введено признака кінця
послідовності
            {
                //Створюємо новий елемент списку
                Double_List* Current = stackalloc
Double_List[sizeof(Double_List)];
                Current->Data = n; //Заносимо значення в
інформаційне поле
                Current->Next = null; //В адресне поле заносимо
0
                Current->Prev = null;

                if (Head == null)  //якщо елемент перший,
                {                  //він становиться головою та
хвостом списку
                    Head = Last = Current;

                }
                else
                {
                    //Додавання елемента на початок списку
                    Current->Next = Head;
                    Head->Prev = Current;
                    Head = Current;
                }
```

```
                //Введення нового елемента
                Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
                n = int.Parse(Console.ReadLine());
                count++;
            }
            Console.WriteLine("Кількість елементів в списку:
{0}", count);
            Console.WriteLine("Виведення елементів  з початку
списку");
            //Виведення елементів  з початку списку
            Double_List* Cur = Head;
            while (Cur != null) //Поки не кінець списку
            {
                Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
                Cur = Cur->Next; //Перехід до наступного
елемента
            }

            Console.WriteLine("Виведення елементів  з кінця
списку");
            //Виведення елементів  з кінця списку
            Cur = Last;
            while (Cur != null) //Поки не кінець списку
            {
                Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
                Cur = Cur->Prev; //Перехід до наступного
елемента
            }

            //включення елемента з заданим номером до
односпрямованого списку
            Console.WriteLine("ВВедіть місце елемента: ");
            int Place = int.Parse(Console.ReadLine());
            Console.WriteLine("ВВедіть елемент: ");
            n = int.Parse(Console.ReadLine());
            //Створюємо новий елемент списку
            Double_List* NewItem = stackalloc
Double_List[sizeof(Double_List)];
            NewItem->Data = n; //Заносимо значення в
інформаційне поле
            NewItem->Next = null; //В адресне поле заносимо 0
```

```
            if (Head == null)  //якщо список порожній,
            {                  //елемент становиться головою та
хвостом списку
                Head = Last = NewItem;
            }
            else //список не порожній
            {
                if (Place == 0) //Якщо потрібно вставити елемент
на перше місце
                {               //новий елемент становиться
головою списка
                    NewItem->Next = Head; //В адресне поле
нового елемента записуємо адресу попередньої голови
                    Head->Prev = NewItem;
                    Head = NewItem; //Новий елемент становиться
першим елементом списку
                }
                else
                {
                    Double_List* Curr = Head;
                    //Цикл для визначення місця вставлення
нового елемента
                    for (int i = 1; i < Place && Curr->Next !=
null; i++)
                    {
                        Curr = Curr->Next;
                    }

                    if (Curr->Next != null) //Вставлення в
середину списка
                    {
                        Curr->Next->Prev = NewItem;
                        NewItem->Next = Curr->Next;
                        Curr->Next = NewItem;
                        NewItem->Prev = Curr;
                    }
                    else     //Вставлення в кінець списка
                    {
                        //Curr->Next = NewItem;
                        Last->Next = NewItem;
                        NewItem->Prev = Last;
                        Last = NewItem;

                    }
```

```
                }
            }
            //Виведення елементів списку
            Cur = Head;
            while (Cur != null) //Поки не кінець списку
            {
                Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
                Cur = Cur->Next; //Перехід до наступного
елемента
            }
            Console.WriteLine("ВВедіть номер  елемента для
виключення: ");
            Place = int.Parse(Console.ReadLine());
            Double_List* Curr1 = Head;
            //Цикл для визначення місця видалення елемента
            for (int i = 1; i < Place && Curr1->Next != null;
i++)
            {
                Curr1 = Curr1->Next;
            }
            if (Curr1 != null) //перевірка на коректність
            {
                if (Curr1 == Head)
                {//виключення першого элемента
                    Head = Head->Next;
                    Head->Prev = null;
                }
                else
                {
                    //виключення останнього елемента
                    if (Curr1->Next==null)
                    {
                        Curr1->Prev->Next = null;
                        Last = Curr1->Prev->Next;
                    }
                    else
                    {//виключення не першого та не останнього
елемента
                        Curr1->Prev->Next = Curr1->Next;
                        Curr1->Next->Prev = Curr1->Prev;
                    }
                }
                //Виведення елементів списку
```

```
                Cur = Head;
                while (Cur != null) //Поки не кінець списку
                {
                    Console.WriteLine(Cur->Data); //Виведення
значення інформаційного поля
                    Cur = Cur->Next; //Перехід до наступного
елемента
                }
                Console.ReadLine();
            }
        }
    }
}
```

Приклад 3.2.2. N – натуральних чисел є елементами двоспрямованого списку L, обчислити: $X_1 * X_n + X_2 * X_{n-1} + ... + X_n * X_1$. Вивести на екран кожний добуток та підсумкову суму.

Алгоритм:

Створюємо структуру.

Формуємо список цілих чисел.

Просуваємося за списком: від початку до кінця і від кінця до початку в одному циклі, перемножуємо дані, що містяться у відповідних елементах списку.

Підсумовуємо отримані результати.

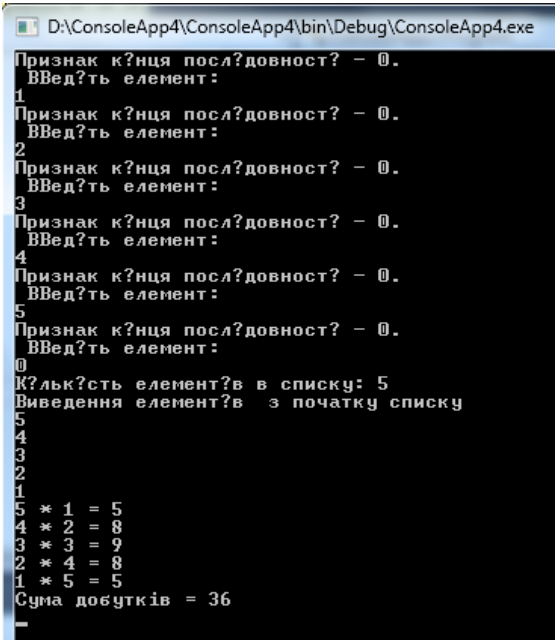
Виводимо на друк.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp2
{
    class Program
    {
        //структура даних для організації роботи з двозв'язним
списком
        public unsafe struct Double_List
        {
            public int Data; //інформаційне поле
```

```
            public Double_List* Next; //адресне поле для зв'язку
с наступним елементом
            public Double_List* Prev; //адресне поле для зв'язку
с попереднімелементом
        };
        static unsafe void Main(string[] args)
        {
            //покажчик на перший елемент списку
            Double_List* Head = null;
            //покажчик на останній элемент списку (за
необхідностю)
            Double_List* Last = null;
            //Введення першого елемента списку
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;  //Лічильник елементів списку
            while (n != 0) //Поки не введено признака кінця
послідовності
            {
                //Створюємо новий елемент списку
                Double_List* Current = stackalloc
Double_List[sizeof(Double_List)];
                Current->Data = n; //Заносимо значення в
інформаційне поле
                Current->Next = null; //В адресне поле заносимо
0
                Current->Prev = null;

                if (Head == null)  //якщо елемент перший,
                {                  //він становиться головою та
хвостом списку
                    Head = Last = Current;
                    Last->Prev = Head;
                }
                else
                {
                    //Додавання елемента на початок списку
                    Current->Next = Head;
                    Head->Prev = Current;
                    Head = Current;
                }
                //Введення нового елемента
```

```
                Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
                n = int.Parse(Console.ReadLine());
                count++;
            }

            Console.WriteLine("Кількість елементів в списку:
{0}", count);
            Console.WriteLine("Виведення елементів  з початку
списку");
            //Виведення елементів  з початку списку
            Double_List* CurN = Head, CurP=Last;
            double Dob = 1;
            //Виведення елементів списку
                while (CurN != null) //Поки не кінець списку
                {
                    Console.WriteLine(CurN->Data); //Виведення
значення інформаційного поля
                    CurN = CurN->Next; //Перехід до наступного
елемента
                }

            CurN = Head;
            while (CurN != null) //Поки не кінець списку
            {
                Console.WriteLine("{0} * {1} = {2}", CurN->Data,
CurP->Data,
                CurN->Data * CurP->Data); //Виведення значення
інформаційного поля
                Dob += CurN->Data * CurP->Data;
                CurN = CurN->Next; //Перехід до наступного
елемента
                CurP = CurP->Prev;
            }
            Console.WriteLine("Сума добутків = {0}", Dob);
            Console.ReadLine();


        }
    }
}
```

Результат роботи програми



3.3. Індивідуальні завдання

Потрібно реалізувати кожне завдання у відповідності з наведеними етапами:

1. вивчити словесну постановку задачі, виділяючи при цьому всі види даних;
2. сформулювати математичну постановку задачі;
3. обрати метод розв’язання задачі, якщо це необхідно;
4. розробити графічну схему алгоритму;
5. записати розроблений алгоритм мовою програмування;
6. розробити контрольний тест до програми;
7. налагодити програму;
8. представити звіт до роботи.

Варіант кожного завдання обирається за номером студента в журналі.

Завдання 1.

1 Створити лінійний двоспрямований список із символів. Видалити зі списку елемент після кожного символу &.

2 Створити лінійний двоспрямований список з цілих чисел. Вставити в список число 66 після кожного елемента з від'ємним значенням.

3 Створити лінійний двоспрямований список з цілих чисел. Вставити в список останній парний елемент після кожного непарного елемента.

4 Створити лінійний двоспрямований список з цілих чисел. Вставити в список число 25 перед кожним елементом з позитивним значенням. Видалити зі списку всі негативні числа.

5 Створити лінійний двоспрямований список з цілих чисел. Вставити в список число 0 перед кожним числом від 2 до 7. Визначити суму чисел більших за 7.

6 Створити лінійний двоспрямований список з цілих чисел. Визначити суму елементів списку зі значеннями більше або рівними 15. Видалити зі списку всі значення, які менше 5.

7 Створити лінійний двоспрямований список з цілих чисел. Видалити зі списку перший парний елемент, що стоїть на непарній позиції.

8 Створити лінійний двоспрямований список з цілих чисел. Видалити зі списку елемент після кожного елемента, рівного 4. Вставити число 0 перед кожним числом 1.

9 Створити лінійний двоспрямований список з цілих чисел. Видалити зі списку перший елемент, більший числа 4. Вставити в список число 10 перед кожним числом, рівним 15.

10 Створити лінійний двоспрямований список з цілих чисел. Видалити зі списку кожен елемент, кратний трьом. Вставити в список число 88 після кожної пари рівних поруч стоять чисел.

11 Створити лінійний двоспрямований список з цілих чисел. Видалити зі списку перший елемент кратний 5. Вставити

число 44 перед кожним числом кратним 7.

12 Створити лінійний двоспрямований список з символів. Видалити зі списку перший елемент, код якого менше 48. Вставити символ % після кожної цифри.

13 Створити лінійний двоспрямований список з символів. Видалити зі списку елемент перед кожним символом ^. Визначити кількість символів * в списку.

14 Створити лінійний двоспрямований список з дійсних чисел. Продублювати в списку перший позитивний елемент (якщо такого немає, залишити список без зміни).

15 Створити лінійний двоспрямований список з дійсних чисел. Обчислити середнє значення елементів списку і вставити число 11 перед кожним числом, більшим за середнє значення.

16 Створити лінійний двоспрямований список з дійсних чисел. Вставити в список число 1.5 після кожного елемента з від'ємним значенням.

17 Створити лінійний двоспрямований список з дійсних чисел. Вставити перший позитивний елемент списку після кожного негативного числа (якщо такого немає, залишити список без зміни).

18 Створити лінійний двоспрямований список з дійсних чисел. Вставити в список число 13.5 після першого елемента зі значенням більшим 2.

19 Створити лінійний двоспрямований список з дійсних чисел. Вставити в список перший негативний елемент перед кожним числом, рівним 20 (якщо таких немає, залишити список без зміни).

20 Створити лінійний двоспрямований список з дійсних чисел. Визначити суму елементів списку зі значеннями більше або рівними 15.

21 Створити лінійний двоспрямований список з дійсних чисел. Визначити середнє значення елементів списку зі значеннями менше або рівними 15. Видалити зі списку елементи, які більше 25.

22 Створити лінійний двоспрямований список з дійсних чисел. Визначити максимальне з елементів списку і вставити його після кожного елемента зі значенням 1.
23 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням 55.
24 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку елемент після кожного елемента з від'ємним значенням.
25 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку перший елемент менший за модулем 5.
26 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням -2. Вставити число 33 в кінець списку.
27 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку всі числа від 2 до 5.Вставити в список число 1.5 після кожного елемента з від'ємним значенням.
28 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням в інтервалі від 10 до 20.
29 Створити лінійний двоспрямований список з дійсних чисел. Видалити зі списку елементи, у яких дробова частина більше 0,5.
30 Створити лінійний двонаправлений список з цілих чисел. Вставити в список число 11 після кожного елемента, рівного 9.

Завдання 2.

1 Створити циклічний двонаправлений список з дійсних чисел. Видалити зі списку елемент перед кожним елементом із значенням 3.
2 Створити циклічний двонаправлений список з цілих чисел. Видалити зі списку перший елемент зі значенням 10.
3 Створити циклічний двонаправлений список з цілих чисел. Видалити зі списку останній елемент зі значенням меншим 15.

4 Створити циклічний однонаправлений список з дійсних чисел. Вставити в список число 2.5 перед кожним елементом з позитивним значенням.

5 Дан покажчик P1 на початок однозв'язного лінійного списку. Перетворити вихідний (однозв'язний) ланцюжок в двузв'язний, в якому кожен елемент пов'язаний не тільки з наступним елементом (за допомогою поля Next), але і з попереднім (за допомогою поля Prev). Поле Prev першого елемента покласти рівним null. Вивести на екран перетворену ланцюжок в зворотному порядку.

6 Дано покажчик P1 на перший елемент непорожньої двозв'язного списку. Продублювати в списку всі елементи з непарними значеннями (нові елементи додавати перед існуючими елементами з такими ж значеннями) і вивести покажчик на перший елемент перетвореного списку.

7 Дано покажчик P1 на перший елемент непорожньої двозв'язного списку. Продублювати в списку всі елементи з непарними значеннями (нові елементи додавати після існуючих елементів з такими ж значеннями) і вивести покажчик на останній елемент перетвореного списку.

8 Дано покажчик P1 на перший елемент двозв'язного списку, що містить не менше двох елементів. Видалити зі списку всі елементи з непарними номерами і вивести покажчик на перший елемент перетвореного списку.

9 Дано покажчик P1 на перший елемент непорожньої двозв'язного списку. Видалити зі списку всі елементи з непарними значеннями і вивести покажчик на перший елемент перетвореного списку (якщо в результаті видалення елементів списку виявиться порожнім, то вивести null).

10 Дано число K> 0 і покажчик P0 на один з елементів непорожньої двозв'язного списку. Перемістити в списку даний елемент на K позицій вперед (якщо після даного елемента знаходиться менш K елементів, то перемістити його в кінець списку). Вивести покажчики на перший і останній елементи перетвореного списку.

11 Дано покажчик P1 на перший елемент непорожньої двозв’язного списку. Перегрупувати його елементи, перемістивши всі елементи з непарними номерами в кінець списку (в тому ж порядку) і вивести покажчик на перший елемент перетвореного списку.
12 Дано два непорожні двозв’язні списка і пов’язані з ними покажчики: PA і PB вказують на перший і останній елементи першого списку, PC – на один з елементів другого. Об’єднати вихідні списки, помістивши все елементи першого списку (в тому ж порядку) після цього елемента другого списку, і вивести покажчики на перший і останній елементи об’єднаного списку.
13 Дано покажчики PX і PY на два різних елемента двозв’язного списку; елемент з адресою PX знаходиться в списку перед елементом з адресою PY, але не обов’язково поруч з ним. Перемістити елементи, розташовані між даними елементами (не включаючи дані елементи) в новий список (в тому ж порядку). Вивести покажчики на перші елементи перетвореного і нового списків. Якщо новий список виявиться порожнім, то пов’язаний з ним покажчик покласти рівним null.
14 Дано покажчики P1 і P2 на перший і останній елементи непорожнього двозв’язного списку, що містить парну кількість елементів. Перетворити список в два циклічних списка, перший з яких містить першу половину елементів вихідного списку, а другий – другу половину. Вивести покажчики PA і PB на два середніх елемента вихідного списку (елемент з адресою PA повинен входити в перший циклічний список, а елемент з адресою PB – у другій).
15 Дано число K> 0 і покажчики P1 і P2 на перший і останній елементи непорожнього двозв’язного списку. Здійснити циклічний зсув елементів списку на K позицій вперед (тобто в напрямку від початку до кінця списку) і вивести покажчики на перший і останній елементи отриманого списку. Для виконання циклічного зсуву перетворити вихідний список в

циклічний, після чого «розірвати» його в позиції, яка відповідає цьому значенню K.

16 Дано покажчики P1, P2 і P3 на перший, останній і поточний елементи двозв'язного списку (якщо список є порожнім, то P1 = P2 = P3 = null). Також дано число N > 0 і набір з N чисел. Описати тип TList – запис з полями First, Last і Current типу PNode (поля вказують відповідно на перший, останній і поточний елементи списку) – і процедуру InsertLast (L, D), яка додає новий елемент зі значенням D в кінець списку L (L – вхідний і вихідний параметр типу TList, D – вхідний параметр цілого типу). Доданий елемент стає поточним. За допомогою цієї процедури додати в кінець вихідного списку даний набір чисел (в тому ж порядку) і вивести нові адреси його першого, останнього та поточного елементів.

17 Дано покажчики P1, P2 і P3 на перший, останній і поточний елементи двозв'язного списку (якщо список є порожнім, то P1 = P2 = P3 = null). Також дано число N > 0 і набір з N чисел. Використовуючи тип TList (див. завдання 16), описати процедуру InsertFirst (L, D), яка додає новий елемент зі значенням D в початок списку L (L – вхідний і вихідний параметр типу TList, D – вхідний параметр цілого типу). Доданий елемент стає поточним. За допомогою цієї процедури додати в початок вихідного списку даний набір чисел (додані числа будуть розташовуватися в списку в зворотному порядку) і вивести нові адреси його першого, останнього та поточного елементів.

18 Дан непорожній двозв'язний список, перший, останній і поточний елементи якого мають адреси P1, P2 і P3. Також дані п'ять чисел. Використовуючи тип TList (див. завдання 16), описати процедуру InsertBefore (L, D), яка вставляє новий елемент зі значенням D перед поточним елементом списку L (L – вхідний і вихідний параметр типу TList, D – вхідний параметр цілого типу). Вставлений елемент стає поточним. За допомогою цієї процедури вставити п'ять даних чисел в вихідний список і вивести нові адреси його першого,

останнього та поточного елементів.

19 Дан непорожній двозв'язний список, перший, останній і поточний елементи якого мають адреси P1, P2 і P3. Також дані п'ять чисел. Використовуючи тип TList (див. завдання 16), описати процедуру InsertAfter (L, D), яка вставляє новий елемент зі значенням D після поточного елемента списку L (L – вхідний і вихідний параметр типу TList, D – вхідний параметр цілого типу). Вставлений елемент стає поточним. За допомогою цієї процедури вставити п'ять даних чисел в вихідний список і вивести нові адреси його першого, останнього та поточного елементів.

20 Створити циклічний двонаправлений список з цілих чисел. Визначити добуток парних значень елементів списку і вставити елемент зі значенням, рівним обчисленому добутку в кінець списку.

3.4. Зміст|вміст| звіту:

1. Прізвище та ім'я студента.
2. Номер і назва лабораторної роботи.
3. Мета роботи.
4. Номер індивідуального завдання. Текст завдання (постановка завдання).
5. Лістинг програми.
6. Скриншот робочої програми.
7. Висновок про засвоєнні знання та вміння.

3.5. Питання для самоперевірки

1. Чи кожен список є зв'язаним? Обгрунтуйте відповідь.
2. У чому відмінність першого елемента двоспрямованого списку від інших елементів цього ж списку?
3. У чому відмінність останнього елемента двоспрямованого списку від інших елементів цього ж списку?
4. Чому при роботі з двоспрямованим списком не обов'язкове позиціонування на перший елемент списку?

5. У чому принципові відмінності виконання додавання (видалення) елемента на першу і будь-яку іншу позиції в двоспрямованому списку?
6. У чому принципові відмінності виконання основних операцій в односпрямованих і двоспрямованих списках?
7. З якою метою в програмах виконується перевірка на порожність односпрямованого (двоспрямованого) списку?
8. З якою метою в програмах виконується видалення односпрямованого (двоспрямованого) списку після закінчення роботи з ним? Як зміниться робота програми, якщо операцію видалення списку не виконувати?

Лабораторна робота № 4. Стеки і черги. Дек. Поняття стеку, черги, деку. Основні операції над елементами: пошук, додавання, видалення елементів. Реалізація на базі лінійного списку та масиву.

Мета роботи: набути навичок створення та обробки стеку, черги та деку.

Послідовність виконання роботи:

1. Ознайомитись із теоретичними відомостями. (*Актуалізація опорних знань*).
2. Виконати програмування програм за поданими прикладами. Результат представити викладачу (*Застосування набутих знань*).
3. Виконати варіант самостійної роботи. (*Закріплення набутих знань*).
4. Оформити звіт на виконану роботу. (*Узагальнення та систематизація набутих знань*).
5. Захист звітів, відповіді на запитання.

4.1. Теоретичні відомості

Стек як окремий випадок лінійного списку. Особливості організації та обробки. У списках доступ до елементів відбувається за допомогою адресації, при цьому доступ до окремих елементів не обмежений. Але існують також і такі спискові структури даних, в яких є обмеження доступу до елементів. Одним з представників таких спискових структур є стековий список або просто стек.

Стек (англ. Stack – стопка) – це структура даних, в якій новий елемент завжди записується в її початок (вершину) і черговий елемент читається також завжди вибирається з її початку (рис. 4.1). У стеках використовується метод доступу до елементів LIFO (Last Input − First Output, «останнім прийшов − першим вийшов»). Найчастіше принцип роботи стека порівнюють зі стопкою тарілок: щоб взяти другу зверху,

потрібно спочатку взяти верхню.

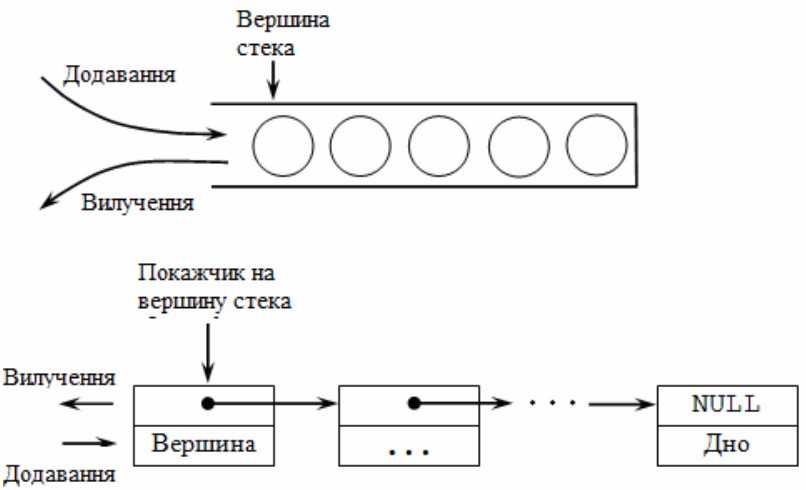


Рис. 4.1. Стек та його організація

Стек має такі властивості:

– елементи додаються у вершину стеку;
– елементи видаляються з вершини стеку;
– покажчик в останньому елементі стеку дорівнює null;
– неможливо вилучити елемент з середини стеку, н вилучивши всі елементи, що йдуть попереду.

Використання стеку в програмуванні:

– під час виклику підпрограм в стеку зберігається адреса повернення до неї;
– стек використовується компілятором під час обчислення виразів;
– в стек записуються значення локальних змінних.

Стек як динамічну структуру даних легко створити на основі лінійного списку. Оскільки робота завжди йде з заголовком стека, тобто не потрібно здійснювати перегляд елементів, видалення і вставку елементів у середину або кінець списку, то досить використовувати економічний по

пам'яті лінійний односпрямований список. Для такого списку досить зберігати покажчик вершини стека, який вказує на перший елемент списку. Якщо стек порожній, то списку не існує, і покажчик набуває значення *null*.

Оголошення елементів стека аналогічно оголошенню елементів лінійного односпрямованого списку. Тому оголосимо стек через оголошення лінійного односпрямованого списку:

```
struct Stack
{
    public int inf;
    unsafe public Stack* next;
}
```

Для роботи зі стеком достатньо мати два покажчики: на голову стеку та допоміжний покажчик на елемент стеку.

Алгоритм вставки елементу до стеку

1. Виділити пам'ять для нового елементу стеку.
2. Ввести дані до нового елементу.
3. Зв’язати допоміжний елемент із вершиною стеку.
4. Встановити вершину теку на новостворений елемент.

Алгоритм видалення елемента із не порожнього стеку

1. Створити копію покажчика на вершину стеку.
2. Перемістити покажчик на вершину стеку на наступний елемент
3. Звільнити пам'ять із-під колишньої вершини стеку

Приклади обробки стеку

//Небезпечний код

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
struct Stack
{
    public int inf;
    unsafe public Stack* next;
```

```
}
class PtrIndexDemo
{
    unsafe static void Main()
    {
        Stack* head = null;
        Console.WriteLine("Ви бажаєте створити стек (у-так)");
        string n = Console.ReadLine();
        int count = 0;

        //Створення стеку
        while (n == "у")
        {
            Stack* current = stackalloc Stack[sizeof(Stack)];
            Console.WriteLine("ВВедіть значення:");
            current->inf = Convert.ToInt32(Console.ReadLine());
            current->next = head;
            head = current;
            count++;
            Console.WriteLine("Ви бажаєте продовжити введення
елементів (у-так)");
            n = Console.ReadLine();
        }
        Console.WriteLine("Кількість елементів в стеку: {0}",
count);
        Stack* cur = head;

        //Перегляд вмісту стеку
        //Стек не знищується
        Console.WriteLine("Елементи стеку виводяться на екран,
але стек не знищується");
        while (cur != null)
        {
            Console.WriteLine(cur->inf);
            cur = cur->next;
        }
        Console.WriteLine((uint)head);

        //Стек знищується
        Console.WriteLine("Елементи стеку виводяться на екран,
стек знищується");
        while (head != null)
        {
            Console.WriteLine(head->inf);
```

```
            head = head->next;
        }
        Console.WriteLine((uint)head);
        Console.ReadLine();
    }
}
```

// Код під керуванням CLR

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
class Stack
{
    public int inf;
    public Stack next;
}
class PtrIndexDemo
{

    static void Main()
    {
        Stack head = null;
        Stack current = null;
        Console.WriteLine("Ви бажаєте створити стек (у-так)");
        string n = Console.ReadLine();
        int count = 0;

        //Створення стеку
        while (n == "у")
        {
            current = new Stack();
            Console.WriteLine("ВВедіть значення:");
            current.inf = Convert.ToInt32(Console.ReadLine());
            current.next = head;
            head = current;
            count++;
            Console.WriteLine("Ви бажаєте продовжити введення
елементів (у-так)");
            n = Console.ReadLine();
        }
```

```
        Console.WriteLine("Кількість елементів в стеку: {0}",
count);

        //Перегляд вмісту стеку
        //Стек не знищується
        current = head;
        while (current != null)
        {
            Console.WriteLine(current.inf);
            current = current.next;
        }


        while (head != null)
        {
            Console.WriteLine(head.inf);
            head = head.next;
        }

        Console.ReadLine();
    }
}
```

Черга як окремий випадок лінійного списку. Особливості організації та обробки. Черга – це один з різновидів однозв'язного лінійного списку. Черга працює за принципом «перший прийшов – перший вийшов» (FIFO – First In First Out). Черга має такі властивості:

- елементи додаються у хвіст черги;
- елементи зчитуються і видаляються з вершини черги;
- покажчик в останньому елементі черги дорівнює null;
- неможливо вилучити елемент з середини стеку, не вилучивши всі елементи, що йдуть попереду.

Використання черг в обчислювальній техніці:

- у мережній операційній системі процесор сервера обслуговує в певний момент часу тільки одного користувача. Запити інших користувачів записуються в чергу. Під час обслуговування користувачів кожен запит просувається до

початку черги. Перший в черзі запит підлягає першочерговому обслуговуванню.

– У комп'ютерній мережі за чергою обслуговуються інформаційні пакети;
– Черги застосовуються для буферизації потоків даних, що виводяться на друк, якщо в комп'ютерній мережі використовується один принтер.

Для роботи зі стеком потрібно мати три покажчики: на голову стеку, на кінець черги та допоміжний покажчик на елемент черги.

Алгоритм вставки елементу до черги

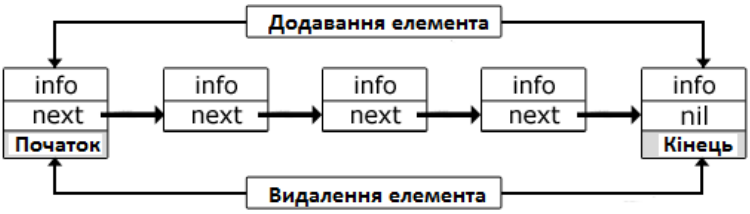
1. Виділити пам'ять для нового елементу стеку.
2. Ввести дані до нового елементу.
3. Вважати новий елемент останнім в черзі.
4. Якщо черга порожня, то ініціалізувати її вершину
5. Якщо черга не порожня, то зв'язати останній елемент із новоутвореним.
6. Вважати новий елемент черги останнім.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
class Queue
{
    public int inf;
    public Queue next;
}
namespace ConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue head = null;
            Queue current = null;
            Queue last = null;
```

```
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;
            //Створення стеку
            while (n != 0)
            {

                current = new Queue();
                count++;
                current.inf = n;
                if (head == null)
                {
                    current.next = head;
                    head = last = current;
                }
                else
                {
                    last.next = current;
                    last = current;
                }


                Console.WriteLine("ВВедіть значення:");
                n = Convert.ToInt32(Console.ReadLine());

            }
            Console.WriteLine("Кількість елементів в черзі:
{0}", count);
            current = head;
            Console.WriteLine("Виведення елементів черги без її
знищення");
            while (current != null)
            {
                Console.WriteLine(current.inf);
                current = current.next;
            }
            Console.WriteLine("Head = {0:X}", head);
            Console.WriteLine("Виведення елементів черги з її
знищенням");

            while (head != null)
            {
                Console.WriteLine(head.inf);
```

```
                head = head.next;
            }
            Console.WriteLine("Head = {0:X}",head);

            Console.ReadLine();
        }
    }
}
```

Дек. Дек – структура даних одночасно працює за двома способам організації даних: FIFO і LIFO. Тому її допустимо віднести до окремої програмної одиниці, отриманої в результаті підсумовування двох попередніх видів структур даних.



Число основних операцій, виконуваних над стеком і чергою, дорівнювало трьом: додавання елемента, видалення елемента, читання елемента. При цьому не вказувалося місце структури даних, активне в момент їх виконання, оскільки раніше воно однозначно визначалося властивостями (визначення) самої структури. Тепер, з огляду на дек як узагальненого випадку, для наведених операцій слід вказати цю область. Розділивши кожну з операцій на дві: одну стосовно «голови» дека, іншу – його «хвоста», отримаємо набір з шести операцій:

1. додавання елемента на початок;
2. додавання елемента в кінець;
3. видалення першого елемента;
4. видалення останнього елемента;
5. читання першого елемента;
6. читання останнього елемента.

На практиці цей список може бути доповнений перевіркою чи дек порожній, отриманням його розміру і деякими іншими операціями.

4.2. Приклади виконання завдань.

Завдання 1.

Створити стек, інформаційними полями якого є: монітор, діагональ і його ціна. Додати в стек відомості про новий монітор. Організувати перегляд даних стека і визначити кількість моніторів з діагоналлю більше 20 дюймів.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
class Stack
{
    public string name;
    public double diag;
    public double cina;
    public Stack next;
}
class PtrlndexDemo
{

    static void Main()
    {
        Stack head = null;
        Stack current = null;
        Console.WriteLine("Ви бажаєте створити стек (у-так)");
        string n = Console.ReadLine();
        int count = 0;

        //Створення стеку
        while (n == "у")
        {
            current = new Stack();
            Console.WriteLine("ВВедіть значення:");
            Console.WriteLine("ВВедіть назву монітора:");
            current.name = Console.ReadLine();
            Console.WriteLine("ВВедіть діагональ монітора:");
```

```
            current.diag = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("ВВедіть ціну монітора:");
            current.cina = Convert.ToDouble(Console.ReadLine());
            current.next = head;
            head = current;
            count++;
            Console.WriteLine("Ви бажаєте продовжити введення
елементів (у-так)");
            n = Console.ReadLine();
        }

        Console.WriteLine("Кількість елементів в стеку: {0}",
count);

        //Перегляд вмісту стеку
        //Стек не знищується
        current = head;
        while (current != null)
        {
            Console.WriteLine(current.name+ "  "+ current.diag+"
"+ current.cina);
            current = current.next;
        }

        //Додавання елемента в стек
        current = new Stack();
        Console.WriteLine("ВВедіть значення:");
        Console.WriteLine("ВВедіть назву монітора:");
        current.name = Console.ReadLine();
        Console.WriteLine("ВВедіть діагональ монітора:");
        current.diag = Convert.ToDouble(Console.ReadLine());
        Console.WriteLine("ВВедіть ціну монітора:");
        current.cina = Convert.ToDouble(Console.ReadLine());
        current.next = head;
        head = current;
        count++;
        Console.WriteLine("Кількість елементів в стеку: {0}",
count);

        //Перегляд вмісту стеку
        //Стек не знищується
        current = head;
        while (current != null)
        {
```

```
            Console.WriteLine(current.name + "   " + current.diag
+ "   " + current.cina);
            current = current.next;
        }
        int Count_Diag = 0;
        //Рахуємо кількість моніторів з діагоналлю >20 дюймів
        while (head != null)
        {
            if (head.diag > 20) Count_Diag++;
             head = head.next;
        }
        Console.WriteLine("Моніторів з діагоналлю більше 20
дюймів: {0}", Count_Diag);
        Console.WriteLine("\nСтек знищено");
        Console.ReadLine();
    }
}
```

```
Ви бажаєте створити стек (y-так)
y
ВВедіть значення:
ВВедіть назву монітора:
sm
ВВедіть діагональ монітора:
12
ВВедіть ціну монітора:
400
Ви бажаєте продовжити введення елементів (y-так)
y
ВВедіть значення:
ВВедіть назву монітора:
hp
ВВедіть діагональ монітора:
45
ВВедіть ціну монітора:
300
Ви бажаєте продовжити введення елементів (y-так)
n
Кількість елементів в стеку: 2
hp   45   300
sm   12   400
ВВедіть значення:
ВВедіть назву монітора:
as
ВВедіть діагональ монітора:
14
ВВедіть ціну монітора:
500
Кількість елементів в стеку: 3
as   14   500
hp   45   300
sm   12   400
Моніторів з діагоналлю більше 20 дюймів: 1

Стек знищено
```

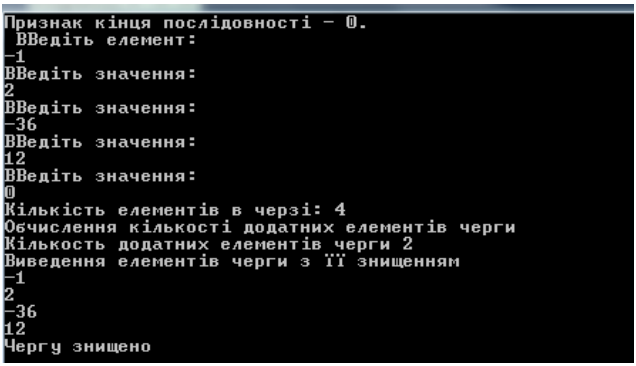
Завдання 2.

Створити чергу з цілих чисел. Визначити кількість додатних елементів черги. Організувати перегляд даних черги.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
class Queue
{
    public int inf;
    public Queue next;
}
namespace ConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue head = null;
            Queue current = null;
            Queue last = null;
            Console.WriteLine("Признак кінця послідовності -
0.\n ВВедіть елемент: ");
            int n = int.Parse(Console.ReadLine());
            int count = 0;
            //Створення стеку
            while (n != 0)
            {

                current = new Queue();
                count++;
                current.inf = n;
                if (head == null)
                {
                    current.next = head;
                    head = last = current;
                }
                else
                {
```

```
                    last.next = current;
                    last = current;
                }


                Console.WriteLine("ВВеділь значення:");
                n = Convert.ToInt32(Console.ReadLine());

            }
            Console.WriteLine("Кількість елементів в черзі:
{0}", count);
            current = head;

            Console.WriteLine("Обчислення кількості додатних
елементів черги");
            int Count_P = 0;
            while (current != null)
            {
                if (current.inf > 0) Count_P++;
                current = current.next;
            }

            Console.WriteLine("Кількость додатних елементів
черги {0}", Count_P);

            Console.WriteLine("Виведення елементів черги з її
знищенням");

            while (head != null)
            {
                Console.WriteLine(head.inf);
                head = head.next;
            }
            Console.WriteLine("Чергу знищено");

            Console.ReadLine();
        }
    }
}
```

Результат роботи програми:



4.3. Індивідуальні завдання

Потрібно реалізувати кожне завдання у відповідності з наведеними етапами:

1. вивчити словесну постановку задачі, виділяючи при цьому всі види даних;
2. сформулювати математичну постановку задачі;
3. обрати метод розв’язання задачі, якщо це необхідно;
4. розробити графічну схему алгоритму;
5. записати розроблений алгоритм мовою програмування;
6. розробити контрольний тест до програми;
7. налагодити програму;
8. представити звіт до роботи.

Варіант кожного завдання обирається за номером студента в журналі.

Завдання 1.

1 Створити стек з цілих чисел. Обчислити добуток непарних значень елементів стека. Організувати перегляд даних стека.

2 Дано число N > 0 і набір з N чисел. Створити стек, що містить вихідні числа (останнє число буде вершиною стека), і вивести покажчик на його вершину.

3 Створити стек з дійсних чисел. Визначити максимальний елементів стеку. Організувати перегляд даних стека.

4 Створити стек, інформаційними полями якого є: прізвище та середній бал студента. Додати в стек відомості про новий студента. Організувати перегляд даних стека.
5 Створити стек, інформаційними полями якого є: назва гори і висота. Додати в стек відомості про нову горе. Організувати перегляд даних стека і визначити середню висоту гір.
6 Створити стек, інформаційними полями якого є: назва книги і кількість сторінок. Додати в стек відомості про нову книгу. Організувати перегляд даних стека і визначити кількість книг в стеці.
7 Створити стек, інформаційними полями якого є: вулиця, номер будинку і номер квартири. Додати в стек відомості про нову квартиру. Організувати перегляд даних стека і визначити кількість будинків на вулиці «Дерибасівська».
8 Створити стек, інформаційними полями якого є: найменування товару і його ціна. Додати в стек відомості про новий товар. Організувати перегляд даних стека і обчислити середню ціну товарів.
9 Створити стек з цілих чисел. Обчислити середнє арифметичне парних значень елементів стека. Організувати перегляд даних стека.
10 Створити стек, інформаційними полями якого є: книга і її ціна. Додати в стек відомості про нову книгу. Організувати перегляд даних стека і обчислити середню ціну книг.
11 Створити стек, інформаційними полями якого є: диск і його обсяг. Додати в стек відомості про новий диск. Організувати перегляд даних стека і обчислити диск з максимальним обсягом.
12 Створити стек, інформаційними полями якого є: прізвище працівника і його оклад. Додати в стек відомості про новий працівника. Організувати перегляд даних стека і обчислити середній оклад.
13 Створити стек, інформаційними полями якого є: монітор, діагональ і його ціна. Додати в стек відомості про новий монітор. Організувати перегляд даних стека і визначити

кількість моніторів з діагоналлю більше 20 дюймів.

14 Створити стек, інформаційними полями якої є: довжини катетів прямокутного трикутника (два дійсних числа). Додати в стек відомості про новий трикутник. Організувати просмотр даних черги. Визначити периметр трикутника останнього в стеку.

15 Створити стек, інформаційними полями якого є: книга і її ціна. Додати в стек відомості про нову книгу. Організувати перегляд даних стеку і обчислити середню ціну книг.

16 Створити стек цілочисельних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента зі стека. Додайте в стек числа 4, 3, 1, 2, 4 і роздрукуйте вміст стека. Видаліть один елемент з стека, і роздрукуйте вміст стека ще раз. Знайдіть мінімальний елемент, що належить стеку.

17 Створити стек рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «abc», «fx», «glc», «hi», «gogo» і роздрукуйте вміст стека. Видаліть один елемент з стека, потім додайте рядок «the end» і роздрукуйте вміст стека ще раз. Знайдіть кількість рядків в стеці, що складаються з 2 символів.

18 Створити стек цілочисельних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек числа 1, 2, 3, 4, 5 і роздрукуйте вміст стека. Видаліть 3 елементи з стека, і роздрукуйте вміст стека ще раз. Знайдіть суму елементів стека.

19 Створити стек рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «sdf», «2», «ssd4», «hello» і роздрукуйте вміст стека. Видаліть 2 елементи з стека, і роздрукуйте вміст стека ще раз. Знайдіть рядок мінімальної довжини, що належить стеку.

20 Створити стек цілочисельних значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек числа -5, 3, -4, 5 і роздрукуйте вміст стека. Видаліть один елемент з стека, додайте число 10 в стек, і надрукуйте вміст стека ще раз. Знайдіть суму всіх позитивних елементів, що належать стеку.

21 Створити стек рядкових значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «Students», «of», «the», «group», «TE» і роздрукуйте вміст стека. Видаліть один елемент з стека, і роздрукуйте вміст стека ще раз. Надрукуйте всі рядки, що починаються з малої літери «t», що належать стеку.

22 Створити стек рядкових значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «abc», «de», «f», «g», «hi», «jk» і роздрукуйте вміст стека. Видаліть один елемент з стека, і роздрукуйте вміст стека ще раз. Знайдіть кількість односимвольних рядків в стеці.

23 Створити стек цілочисельних значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек числа -5, 3, -4, 5 і роздрукуйте вміст стека. Видаліть один елемент з стека, додайте в стек число 10, і роздрукуйте стек ще раз. Знайдіть суму всіх позитивних елементів, що належать стеку.

24 Створити стек рядкових значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «Students», «of», «the», «group», «TE», «3» і роздрукуйте вміст стека. Видаліть один елемент з стека, і роздрукуйте вміст стека ще раз. Надрукуйте всі рядки, які складаються з двох символів з стека.

25 Створити стек цілочисельних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек числа 1, 4, 2 і роздрукуйте вміст стека. Додайте число 4 в стек, і роздрукуйте вміст стека ще раз. Знайдіть кількість чисел, більших числа 3, що належить стеку.

26 Створити стек рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «111», «2», «sdf4», «bye» і роздрукуйте вміст стека. Видаліть 1 елемент з стека, і роздрукуйте вміст стека ще раз. Знайдіть рядок максимальної довжини, що належить стеку.

27 Створити стек цілочисельних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек числа 5, 3, 44, 555 і роздрукуйте вміст стека. Видаліть 2 елементи з стека, додайте числа 20 і 30 в стек, і надрукуйте вміст стека ще раз. Знайдіть кількість позитивних двозначних чисел, що належать стеку.

28 Створити стек рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек рядки «Student», «of», «the», «OSAT» і роздрукуйте вміст стека. Видаліть один елемент з стека, додайте рядок «ONAT» і роздрукуйте вміст стека ще раз. Порахуйте кількість рядків, що складаються не менше ніж з трехсімволов, що належать стеку.

29 Створити стек цілочисельних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додайте в стек числа 1, 2, -3, 4 і роздрукуйте вміст стека. Видаліть один елемент з стека, і роздрукуйте вміст стека ще раз. Знайдіть максимальний елемент, що належить стеку.

Завдання 2.

1 Створити чергу з дійсних чисел. Визначити кількість додатних значень елементів черги. Організувати перегляд даних черзі.
2 Створити чергу, інформаційними полями якого є: комп'ютер і обсяг його оперативної пам'яті. Видалити з черги відомості про комп'ютер, які були введені першими. Організувати перегляд даних черзі і обчислити загальний обсяг пам'яті комп'ютерів, записаних в черзі.
3 Дано число D і покажчики P1 і P2 на початок і кінець черги, що містить не менше двох елементів. Додати елемент зі значенням D в кінець черги і витягти з черги перший (початковий) елемент. Вивести значення витягнутого елемента і вміст черги
4 Створити чергу, інформаційними полями якої є: телефон і його ціна. Видалити з черги дані про телефон, які були введені першими. Організувати перегляд даних черги.
5 Створити чергу, інформаційними полями якої є: прізвище та середній бал студента. Додати в чергу відомості про новий студента. Організувати перегляд даних черзі.
6 Створити чергу, інформаційними полями якої є: довжини катетів прямокутного трикутника (два дійсних числа). Додати в чергу відомості про новий трикутник. Організувати просмотр даних черги. Визначити периметр трикутника на початку черги.
7 Дано число N > 0 і покажчики P1 і P2 на початок і кінець непорожній черги. Витягти з черги N початкових елементів і вивести їх значення (якщо чергу містить менше N елементів, то витягти всі її елементи).
8 Створити чергу з дійсних чисел. Визначити мінімальний елемент черги. Організувати перегляд даних черги.
9 Створити чергу, інформаційними полями якої є: найменування товару та його вартість. Додати в чергу відомості про новий товар. Організувати перегляд даних черги і обчислити загальну вартість товарів з

найменуванням «Ручка кулькова».

10 Створити чергу, інформаційними полями якої є: найменування процесора і його тактова частота і кількість ядер. Додати в чергу відомості про новий процесор. Організувати перегляд даних черги і роздрукувати дані про багатоядерні процесори (кількість ядер більше 1).

11 Створити чергу з цілих чисел. Визначити кількість парних значень елементів черги. Організувати перегляд даних черги.

12 Створити чергу з цілих чисел. Визначити середнє значення елементів черги. Організувати перегляд даних черги.

13 Створити чергу з цілих чисел. Визначити кількість додатних елементів черги. Організувати перегляд даних черги.

14 Створити чергу, інформаційними полями якого є: книга і її ціна. Додати в чергу відомості про нову книгу. Організувати перегляд даних черги і обчислити середню ціну книг.

15 Створити чергу з відомостей про клієнтів банку: прізвища та суми на рахунку. Визначити кількість клієнтів банку, у яких сума на рахунку більше 10000 гр. Організувати перегляд даних черги.

16 Створити чергу з цілих чисел. Визначити кількість елементів черги менших 10. Організувати перегляд даних черги.

17 Створити чергу з дійсних чисел. Визначити кількість від’ємних чисел черги. Організувати перегляд даних черги.

18 Створити чергу дійсних значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа: -2.2, 2.3, 2.2, 5.1, 6.7 і роздрукуйте вміст черги. Видаліть 3 елементи з черги, потім додайте в чергу число 1.9 і роздрукуйте чергу ще раз. Знайдіть добуток елементів, що належать черзі.

19 Створити чергу рядкових значень, для реалізації використовуючи однозв’язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four» і

роздрукуйте вміст черги. Видаліть 2 елементи з черги, потім додайте в чергу рядок «inf» і роздрукуйте чергу ще раз. Знайдіть сумарну довжину рядків, що належать черзі, крім першого рядка черги.

20 Створити чергу дійсних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа 2.1, 2.1, 5.3 і роздрукуйте вміст черги. Видаліть 1 елемент з черги, потім додайте в чергу число 4.9 і роздрукуйте чергу ще раз. Знайдіть суму елементів черги.

21 Створити чергу дійсних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа 2.2, 1.2, 2.0, 5.2 і роздрукуйте вміст черги. Видаліть 2 елементи з черги, потім додайте в чергу число 2.9 і роздрукуйте чергу ще раз. Знайдіть суму елементів черги.

22 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four» і роздрукуйте вміст черги. Видаліть 1 елемент з черги, потім додайте в чергу рядок «five» і роздрукуйте чергу ще раз. Знайдіть сумарну довжину всіх рядків, що належать черзі.

23 Створити чергу дійсних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа 2.2, 3.2, 2.4, -3.2 і роздрукуйте вміст черги. Видаліть 1 елемент з черги, потім додайте в чергу число 0.04 і роздрукуйте чергу ще раз. Знайдіть суму чисел по модулю менших 1, що належать черзі.

24 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four», «five»,

«six», «seven» і роздрукуйте вміст черги. Видаліть 1 елемент з черги, потім додайте в чергу рядок «eight» і роздрукуйте чергу ще раз. Знайдіть кількість рядків починаються з літер «s» або «t».

25 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four» і роздрукуйте вміст черги. Видаліть 1 елемент з черги, потім додайте в чергу рядок «five» і роздрукуйте чергу ще раз. Знайдіть сумарну довжину рядків, що належать черзі.

26 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа 46.5, 3.4, 32.4, -3.21 і роздрукуйте вміст черги. Видаліть 2 елементи з черги, потім додайте в чергу число 5.0 і роздрукуйте чергу ще раз. Знайдіть суму елементів, по модулю великих 12, що належать черзі.

27 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four», «five», «six», «seven» і роздрукуйте вміст черги. Видаліть 4 елементи з черги, потім додайте в чергу рядки «eight», «nine» і роздрукуйте чергу ще раз. Знайдіть кількість рядків, що складаються з 4 символів.

28 Створити чергу дійсних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа -2.1, 1.3, -1.34, 3.3 і роздрукуйте вміст черги. Видаліть 1 елемент з черги, потім додайте в чергу число 2.9 і роздрукуйте чергу ще раз. Знайдіть суму від'ємних елементів черги.

29 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції

додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four» і роздрукуйте вміст черги. Додайте в чергу рядок «five» і роздрукуйте чергу ще раз. Знайдіть сумарну довжину всіх рядків, що належать черзі, крім останнього рядка черги.

30 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four» і роздрукуйте вміст черги. Додайте в чергу рядок «five» і роздрукуйте чергу ще раз. Знайдіть сумарну довжину всіх рядків, що належать черзі, крім останнього рядка черги.

31 Створити чергу дійсних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа -2.2, 5.5, 4.3, -4.5 і роздрукуйте вміст черги. Видаліть 1 елемент з черги і роздрукуйте чергу ще раз. Знайдіть суму чисел по модулю більше 4, що належать черзі.

32 Створити чергу рядкових значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу рядки «one», «two», «three», «four», «five», «six», і роздрукуйте вміст черги. Видаліть 2 елементи з черги, потім додайте в чергу рядок «seven» і роздрукуйте чергу ще раз. Знайдіть кількість рядків починаються з літер «f» або «t».

33 Створити чергу дійсних значень, для реалізації використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додайте в чергу числа -2.0, 2.0, 2.1, -2.1, 3.7 і роздрукуйте вміст черги. Видаліть 2 елементи з черги, потім додайте в чергу число 1.1 і роздрукуйте чергу ще раз. Знайдіть добуток додатних елементів, що належать черзі.

34 Дано дві непусті черги; адреси початку і кінця першої рівні

P1 і P2, а друг – P3 і P4. Черги містять однакову кількість елементів. Об'єднати черги в одну, в якій елементи вихідних черг чергуються (починаючи з першого елемента першої черги). Вивести покажчики на початок і кінець отриманої черзі. Операції виділення пам'яті не використовувати.

Завдання 3 (підвищенної складності).

1 Дано дві непорожні черги; адреси початку і кінця першої P1 і P2, а другої – P3 і P4. Елементи кожної з черг впорядковані за зростанням (в напрямку від початку черги до кінця). Об'єднати черги в одну зі збереженням упорядкованості елементів. Вивести покажчики на початок і кінець отриманої черги. Операції виділення пам'яті не використовувати, поля з даними (Data) не pvsy.dfnb.

2 Арифметичний вираз можна представити в зворотної польської записи, де знаки операції слідують за операндами (а не ставляться між ними, як у звичайній запису виразів). Зворотній польський запис не вимагає дужок. Наприклад, висловом «1 + 2» відповідає запис «1 2+», висловом «1 + 2 * 3» запис «1 2 3 * +» (спочатку множаться 2 на 3, а потім 1 складається з результатом), «(2 + 3) * (3 - 1) »записується як« 2 3 + 3 1 - * ». Дано рядок-вираз в зворотної польської записи (числа і знаки +, -, * розділені пробілами). Використовуючи стек, обчисліть значення виразу. **Підказка:** потрібно послідовно перебрати всі числа і знаки з рядка, числа потрібно заносити в стек, а як зустрінеться знак операції, виймати 2 числа зі стеку, застосовувати до них поточну операцію, а результат заносити в стек.

3 Дано послідовність дужок виду «(», «)», «[», «]», «{», «}». Правильними дужковими послідовностями називаються порожня послідовність, а також «(P)», «[P]», «{P}», де P – деяка правильна послідовність. Наприклад «{} () []» і «{[] [() ()]} ()» - правильні дужкові послідовності, а «(]», «[()» і «({)}» - неправильні. Визначте, чи заданий рядок є правильним дужковим виразом. **Підказка:** обробіть по черзі всі символи вхідного рядка, поміщаючи відкривають дужки в стек, а для

закривають дужок виймайте відкриваючу дужку зі стека і перевіряйте, чи відповідають вони один одному.

4 Реалізуйте чергу, використовуючи два стека. **Підказка:** стек інвертує порядок елементів (тобто якщо додати 1 2 3 4 5, виймаючи елементи, отримаємо 5 4 3 2 1), тому якщо елементи пройдуть два стека, то відновиться їх вихідний порядок. При записуванні в чергу розміщуйте елемент в перший стек, при добуванні елементу з черги, виймайте елемент з другого стека, а якщо при цьому другий стек виявиться порожнім, перенесіть все елементи з першого стека в другій (діставайте елементи першого стека, поки вони є, і переносите в другій).

5 Дано число N>0 і дві непорожні черги; адреси початку і кінця першої P1 і P2, а другий - P3 і P4. Перемістити N початкових елементів першої черги в кінець другої черги. Якщо перша черга містить менше N елементів, то перемістити з першої черги в другу все елементи. Вивести нові адреси початку і кінця першої, а потім другої черги (для порожній черзі двічі вивести null). Операції виділення пам'яті не використовувати.

6 Дано набір з 10 чисел. Створити дві черги: перша повинна містити числа з вихідного набору з непарними номерами (1, 3, ..., 9), а друга - з парними (2, 4, ..., 10); порядок чисел в кожній черзі повинен збігатися з порядком чисел в початковому наборі. Вивести покажчики на початок і кінець першої, а потім другої черги.

7 Дано набір з 10 чисел. Створити дві черги: перша повинна містити всі непарні, а друга - всі парні числа з вихідного набору (порядок чисел в кожній черзі повинен збігатися з порядком чисел в початковому наборі). Вивести покажчики на початок і кінець першої, а потім другої черги (одна з черг може виявитися марною; в цьому випадку вивести для неї дві константи null).

8 Дано покажчики P1 і P2 на початок і кінець непорожньої черги. Витягувати з черги елементи, поки значення початкового елемента черги не стане парним, і виводити

значення витягнутих елементів (якщо черга не містить елементів з парними значеннями, то витягти всі її елементи). Вивести також нові адреси початку і кінця черги (для порожньої черги двічі вивести null).

9 Дано дві черги; адреси початку і кінця першої P1 і P2, а другої - P3 і P4 (якщо черга є порожньою, то відповідні адреси рівні null). Перемістити всі елементи першої черги (в порядку від початку до кінця) в кінець другої черги і вивести нові адреси початку і кінця другої черги. Операції виділення пам'яті не використовувати.

10 Дано дві непорожні черги; адреси початку і кінця першої P1 і P2, а другої - P3 і P4. Переміщати елементи з початку першої черги в кінець другої, поки значення початкового елемента першої черги не стане парним (якщо перша черга не містить парних елементів, то перемістити з першої черги в другу все елементи). Вивести нові адреси початку і кінця першої, а потім другої черги (для порожньої черги двічі вивести null). Операції виділення пам'яті не використовувати.

4.4. Зміст|вміст| звіту:

1. Прізвище та ім’я студента.
2. Номер і назва лабораторної роботи.
3. Мета роботи.
4. Номер індивідуального завдання. Текст завдання (постановка завдання).
5. Лістинг програми.
6. Скриншот робочої програми.
7. Висновок про засвоєнні знання та вміння.

4.5. Питання для самоперевірки

1 У чому переваги і недоліки організації структур у вигляді стека?

2 У чому переваги і недоліки організації структур у вигляді черги?

3 Для моделювання яких реальних завдань зручно використовувати стек? А для яких чергу?

4 Яке значення зберігає покажчик на стек?

5 Яке значення зберігає покажчик на чергу?

6 Які існують обмеження на тип інформаційного поля стеки і черги?

7 З якою метою в програмах виконується перевірка на порожність стека і черги?

8 При роботі зі стеком або чергою доступні позиції обмеженого числа елементів. Чи можлива ситуація запису нових елементів стека або черги на вже зайняті власними елементами ділянки пам’яті (запис поверх себе)? Відповідь обґрунтуйте.

Рекомендована література

1. Альфред Ахо. Структуры данных и алгоритмы. М. : Вильямс, 2007. 400 с.
2. Вирт Н. Алгоритмы и структуры данных. М., 2012. 272 с.
3. Седжвик Н. Фундаментальные алгоритмы на С++. Части 1-4.
4. Диасофт, 2001. 688 с.
5. Кнут Д. Искусство программирования. Т. 1-4. М. : Вильямс, 2006. 682 с.
6. Алгоритми і структури даних: конспект лекцій. Частина 1. Структури даних / Укладачі: О. Д. Воробйова, Л. В. Глазунова. Одеса : ОНАЗ ім. О. С. Попова, 2017. 48 с.