# Програмне забезпечення об'єктів Кіберпростору.

# Мова програмування С

Міністерство освіти і науки України
Чернівецький національний університет
імені Юрія Федьковича

# Програмне забезпечення об’єктів Кіберпростору. Мова програмування С

Методичні рекомендації до лабораторних робіт

Укладачі: Галюк С.Д., Круліковський О.В

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2022

УДК 004.4(076.5)+004.43(076.5)
П-784

Друкується за ухвалою редакційно-видавничої ради
Чернівецького національного університету
імені Юрія Федьковича

Укладачі: Галюк С.Д., Круліковський О.В

П-784 **Програмне** забезпечення об’єктів кіберпростору. Мова програмування С : метод. реком. до лабор. робіт. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2022. 100 с.

*Лабораторний практикум із об'єктозорієнтованого програмування мовою С відповідає навчальному плану дисципліни нормативної підготовки “Програмне забезпечення об'єктів кіберпростору” для студентів бакалаврату спеціальності 125 – “Кібербезпека”.*

**УДК 004.4(076.5)+004.43(076.5)**

# Програмне забезпечення об'єктів кіберпростору. Мова програмування С

Методичні рекомендації до лабораторних робіт

Укладачі: Галюк С.Д., Круліковський О.В.

29 вересня 2022 р.

# Зміст

Вступ . . . . . 4
Лабораторна робота 1. Програмування лінійних алгоритмів . . . . . 6
Лабораторна робота 2. Оператори галуження та циклу . . . . . 23
Лабораторна робота 3. Масиви . . . . . 33
Лабораторна робота 4. Використання підпрограм користувача . . . . 41
Лабораторна робота 5. Символьні дані . . . . . 50
Лабораторна робота 6. Робота зі структурами та файлами . . . . . 59
Лабораторна робота 7. Динамічна пам'ять . . . . . 71
Лабораторна робота 8. Сортування і пошук . . . . . 80
Лабораторна робота 9. Динамічні структури даних . . . . . 93
Список літератури . . . . . 102

# Вступ

Програмний захист інформаційних ресурсів займає ключову позицію при побудові комплексних систем безпеки. Для вивчення програмування недостатньо слухати лекції чи читати літературу, основним засобом навчання є розв'язування власних задач із написанням відповідних програми. Це дає змогу глибше зрозуміти як суть алгоритмізації, так і особливості застосування синтаксису мови програмування.

Мова С була створена співробітником американської корпорації Bell Laboratory Денісом Рітчі на початку 70-х років. Протягом тривалого періоду ця мова залишається однією з основних універсальних мов програмування. Серед переваг мови С відзначають переносність програм на комп'ютери різної архітектури та з однієї операційної системи в іншу, лаконічність подання алгоритмів, можливість написати ефективний і швидкий код програми. Широка сфера використання мови С зумовлена ще й тим, що вона є мовою високого рівня, яка підтримує модульність, блокову структуру програм, високий рівень абстракцій, широкі можливості розширення системи програмування та її адаптації до будь-якої предметної області, можливість роздільної компіляції модулів. Водночас, мова має набір низькорівневих засобів, що дають змогу працювати з апаратними засобами комп'ютера і зовнішніми пристроями на побітовому рівні. Мова С – одна з небагатьох стандартизованих мов програмування. Останнім прийнятим стандартом є ISO/IEC 9899:2018.

Мова С лежить в основі створення і розвитку низки сучасних мов об'єктоорієнтованого програмування, таких як C++, Java, C#, JavaScript, Python, Rust, Verilog та інших. Розширення мови С – мова C++ вивчається у другій частині курсу.

Дисципліна "Програмне забезпечення об'єктів кіберпростору" є базовим курсом з підготовки бакалавра зі спеціальності 125 – "Кібербезпека", і цим забезпечує опанування студентом сучасних технологій аналізу та розробки програмного забезпечення.

Мета лабораторного практикуму – засвоєння студентами принципів процедурного програмування та набуття навичок їх реалізації при розробленні додатків мовою програмування С.

Лабораторний практикум передбачає на написання консольних програми. Це зумовлено двома причинами. З одного боку, саме консольні програми є базовим видом прикладних програм, з іншого – таке обмеження дає змогу у процесі занять краще зосередитися саме на питаннях власне програмування, не обтяжуючи себе необхідністю вникати в деталі створення елементів графічного інтерфейсу.

Під час вивчення першої частини курсу "Програмне забезпечення об'єктів кіберпростору" передбачається, що студенти успішно оволодіють основними поняття мови C: типи даних, змінні, константи, логічні та арифметичні операції, умовні та циклічні оператори, структури, макроси, модульність, вказівник та динамічний розподіл пам'яті тощо.

Протягом першої частини курсу студенти повинні виконати дев'ять лабораторних робіт, що відповідають ряду основних тем лекційних занять. До кожної роботи наведені короткі теоретичні відомості з прикладами програм, покликані допомогти у виконанні основних завдань. Для написання коду та компіляції програми рекомендується використовувати інтегроване середовище розробника Code::Blocks IDE або Visual Studio.

# Лабораторна робота 1
# Програмування лінійних алгоритмів

**Мета роботи:** ознайомити з основними поняттями та структурою мови програмування С, базовими типами даних, навчити використовувати бібліотечні функції *введення/виведення* та програмувати лінійні алгоритми.

## Теоретичні відомості

У будь-якій природній мові можна виокремити структурні елементи: символи, слова, словосполучення та речення. В алгоритмічних мовах програмування існують аналогічні структурні одиниці, тільки з іншими назвами: слова називають лексемами, словосполучення – виразами, речення – операторами. Лексеми у свою чергу утворюються із символів, вирази – із лексем і символів, оператори – із символів, лексем і виразів.

*Алфавіт мови програмування*, або її *символи* – це неподільні знаки, за допомогою яких пишуть код програми.

*Лексема*, або *елементарна конструкція* – мінімальна одиниця мови, яка має самостійний зміст.

*Вираз* задає правило обчислення деякого значення.

*Оператор* позначає деяку дію.

Алфавіт мови С включає:

- великі та малі літери латинської абетки;
- арабські цифри;
- пробіл, символи табуляції, символ переходу на наступний рядок тощо;
- символи: , . ; : ? ! ’ ” | / \ ∼ _ ∧ ( ) { } [ ] < > # % & − = +∗.

*Ідентифікатори* (*імена*) використовуються для позначення різних об’єктів: змінних, констант, міток, функцій тощо. Для запису ідентифікаторів можна використовувати великі та малі літери латинської абетки, арабські цифри та символ підкреслення. Ідентифікатор не може починатися з цифри і містити пробіли.

Компілятор мови С розглядає літери верхнього та нижнього регістрів як різні символи. Тому можна створювати ідентифікатори, які збігаються орфо-

графічно, але відрізняються регістром літер. Наприклад, кожен із цих ідентифікаторів унікальний:

`Tim` `tim` `tIm` `TIM` `tIM` `tiM`

*Ключові слова* – набір зарезервованих ідентифікаторів, значення або зміст яких не можна змінювати. Ключові слова мають спеціальне значення для компілятора, їх використання чітко регламентоване. Необхідно пам'ятати, що ідентифікатори, введені програмістом, не повинні збігатися з ключовими словами. До ключових слів мови C належать:

```
auto      break     case      char
const     continue  default   do
double    else      enum      extern
float     for       goto      if
int       long      register  return
short     signed    sizeof    static
struct    switch    typedef   union
unsigned  void      volatile  while
```

*Константами* називають сталі величини, значення яких у процесі виконання програми не змінюються. На відміну від констант, *змінні* можуть набувати іншого значення під час роботи програми. Імена змінних, констант, міток, типів тощо не можуть збігатися із ключовими словами.

Змінні та константи бувають різного типу. Тип даних визначає множину значень та перелік дій або операцій, які можна виконати над кожною змінною даного типу. Вважається, що змінна або вираз належить до конкретного типу, якщо їх значення належить до області допустимих значень цього типу. Тип даних також задає розмір пам'яті, необхідний для збереження змінної чи константи цього типу.

У мові C розрізняють базові типи даних, перелік, пустий тип та похідні (складні) типи. Кожен тип даних позначається власним ідентифікатором, що є одним із ключових слів, наприклад, `int`, `double`, `char`, `void`, ... .

Базові типи даних у C:

- `int` – цілі числа розміром 2 або 4 байти. Як різновиди цілих чисел, у деяких версіях компіляторів існують `short` – коротке ціле (1 байт) та `long` – довге ціле (4 байти);

- `float` – дійсне число з плаваючою комою одинарної точності (4 байти);
- `double` – дійсне число з плаваючою комою подвійної точності (8 байтів);
- `char` – символьний тип даних, використовується для зберігання літер, цифр або інших символів із множини символів ASCII. Значенням об'єкта типу `char` є код символу. Тип char інтерпретується як однобайтове ціле з областю значень від -128 до 127.

Цілі типи `char`, `short`, `int`, `long` можуть використовуватися з модифікаторами `signed` (із знаком) та `unsigned` (без знаку). На відміну від знакових цілих (`signed`), цілі без знака (`unsigned`) не можуть набувати від'ємних значень. Завдяки цьому дещо розширюється діапазон можливих додатних значень типу. Вплив модифікаторів для деяких типів наведено в табл. 1.

Табл. 1: Діапазони значень та розмір деяких типів даних

| **Тип** | **Діапазон значень** | **Розмір (байт)** |
|---|---|---|
| `char` | -128 ... 127 | 1 |
| `short int` | -32768 ... 32767 | 2 |
| `int` | -2 147 483 648 ... 2 147 483 647 | 4 |
| `long int` | $\pm 9.22 * 10^{18}$ | 8 |
| `unsigned char` | 0 ... 255 | 1 |
| `unsigned short` | 0 ... 65535 | 2 |
| `unsigned long` | $0 \ldots 1.84 * 10^{19}$ | 8 |
| `float` | $\pm(1.175 * 10^{-38} \ldots 3.4 * 10^{38})$ | 4 |
| `double` | $\pm(2.225 * 10^{-308} \ldots 1.798 * 10^{308})$ | 8 |
| `long double` | $\pm(3.362 * 10^{-4932} \ldots 1.19 * 10^{4932})$ | 10 |

Щоб використовувати у програмі змінну або константу, їх спочатку необхідно оголосити. Синтаксис оголошення змінної `variable_name` типу `type` такий:

```
type variable_name;
```

Команда оголошення змінної або константи обов'язково повинна закінчуватися крапкою з комою. Якщо ж крапка з комою відсутні, то компілятор покаже помилку.

Для присвоєння змінній значення `value`, його вказують після знака `=`:

```
type variable_name = value;
```

Оголошення змінної `a` цілочислового типу `int`:

```
int a;
```

При такому оголошенні значення змінної `a` невизначене.

Оголосити кілька змінних одного типу можна однією командою, розділивши їх комами, наприклад:

```
int year = 2022, date;
```

Тут оголошено змінну `year` типу `int`, яка дорівнює 2022, та змінну `date` того ж типу.

Ціла та дробова частини дійсних чисел розділяються крапкою:

```
double f = 40.5;
```

Значення символьного типу задається в одинарних лапках. Наприклад, команда створення змінної `c` типу `char`, яка зберігає символ 'A', описується так:

```
char c = 'A';
```

Щоб надати змінній іншого значення, необхідно вказати ім'я змінної і після оператора присвоєння "=" вказати нове значення:

```
int a = 12;
a = 25;
```

Тут після створення змінної `a` її значення дорівнює 12. Після виконання другої команди воно зміниться на 25.

Константи оголошують за допомогою ключового слова `const` перед іменем типу:

```
const double d = 400.5;
const int integer = 128;
const float Pi = 3.14;
const char letter = 'B';
```

При спробі змінити значення константи виникне помилка компіляції:

```
const int a = 12;
a = 25;//тут буде помилка, тому що a - це константа
```

Щоб поліпшити розуміння програми, код супроводжують *коментарями*. У мові С коментарі можуть бути однорядковими та багаторядковими.

Однорядкові коментарі пишуть у чистому рядку або після команди, відділяючи їх знаком "подвійний слеш" – `//`.

Текст на C, що міститься у дужках `/* ... */`, ігноруватиметься компілятором, тобто вважатиметься коментарем до програми. Такі коментарі можуть розміщуватися в будь-якому місці програми і мати довільний розмір:

```
//це коментар
const int a = 12; //це також коментар
/*А це дуже великий коментар
про життя-буття */
```

### Найпростіша програма

*Функцією* в програмуванні називають частину програмного коду, що має власне ім'я, виконує певні дії та може повертати результат. Програма, написана мовою C, містить мінімум одну функцію, яка називається `main`. Робота програми розпочинається з виконання цієї функції. Коректне завершення програми також забезпечується `main`.

Мінімальна програма мовою C має такий вигляд:

```
int main()
{
    return 0;
}
```

Програма містить функцію `main`, яка виконує одну дію: повертає ціле число 0 за допомогою оператора `return`. Усередині `main` (у фігурних дужках) записують команди, які необхідно виконати.

Напишемо програму, яка виводить на екран фразу "Hello, World!". Для цього використаємо функцію `printf`. Щоб розв'язати цю задачу найпростішим способом, необхідно у `printf` передати задану фразу як аргумент. Програма буде такою:

```
1 int main()
2 {
3     printf("Hello, world!");
4     return 0;
5 }
```

### Функції введення та виведення

У стандарті мови С є ряд засобів, які програміст за потреби може використовувати у власних програмах. У більшості випадків такі засоби згруповані за задачами, у яких можуть бути використані.

Функції для введення і виведення інформації описані у файлі *stdio.h*. Тому для їх використання цей файл необхідно підключити до основної програми. Підключення бібліотек виконується командою `#include`, після якої у трикутних дужках вказують файл із розширенням `.h`:

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

Додавати бібліотеки необхідно на прочатку програми перед функцією `main`.

Розглянемо основні функції введення та виведення даних. Сигнатури та призначення функцій такі:

- `int getchar (void)` – введення одного символу;
- `int putchar(int character)` – виведення одного символу.
- `int puts(const char *str)` – виведення рядка (послідовності символів);
- `int scanf(const char *format, arg-list)` – форматоване введення довільних даних;
- `int printf(const char *format, arg-list)` – форматоване виведення довільних даних;

Функція `getchar` повертає один, прочитаний із клавіатури символ, який можна зберегти як значення змінної. Якщо символ прочитати не вдалося, то повертається спеціальне значення `EOF`. Функція `putchar` виводить символ `character` на екран. Приклад використання `getchar` та `putchar`:

```
1  #include <stdio.h>
2  int main()
3  {
4      char ch;//оголосили змінну ch типу char
5      ch = getchar();//зчитати значення одного символу введеного
   ↪   з клавіатури і записати його у змінну ch
6      putchar(ch);//вивести значення ch на екран
7      return 0;
8  }
```

Функція `puts` виводить рядок на екран

```
puts("Hello, Will!");
```

Результатом буде

```
Hello, Will!
```

Після виведення рядка `puts` переводить курсор на наступний, тобто виведення нової інформації почнеться з нового рядка:

```
puts("Hello, Will!");
puts("Hello, Bill!");
```

На екрані отримаємо

```
Hello, Will!
Hello, Bill!
```

Для введення та виведення більш складної інформації використовуються функції `scanf` та `printf`. Першим аргументом функції `printf` є рядок `format`, який задає правило виведення даних, другим `arg-list` – список даних. Функція `printf` перетворює значення аргументів до вигляду, поданого у рядку формату, "збирає" перетворені значення в цей рядок і виводить одержану послідовність символів у стандартний потік виводу.

Для виведення простого рядка функцію `printf` можна використовувати так само, як `puts`:

```
printf("It's hot outside!");
```

Відмінність буде в тому, що `printf` самостійно не переводить курсор на наступний рядок та працює повільніше, ніж `puts`.

Для виведення даних інших типів використовуються специфікатори форматування. Нехай задана цілочислова змінна a, що дорівнює 12. Необхідно вивести її значення на екран. Код програми можна написати так:

```
    int a = 12;
    printf("Значення a дорівнює %d\n",a);
    a = a + 1;
    printf("А зараз це значення більше, a =  %d\n",a);
```

Результат виконання коду:

```
Значення a дорівнює 12
А зараз це значення більше, a =  13
```

Символи `%d` у першому параметрі функції `printf` означають, що ми хочемо вивести ціле число, яке задане як другий параметр `a`. `%d` – це специфікатор форматування, який вказує препроцесору, що на це місце необхідно поставити ціле число. Кількість специфікаторів у рядку форматування повинна дорівнювати кількості аргументів.

Позначення `\n` – команда переведення курсора на новий рядок.

Інші специфікатори форматування, які використовуються функціями форматованого введення-виведення:

- `%c` – символ типу char;
- `%d`, `%i` – десяткове число цілого типу `int`;
- `%f` – дійсне число із плаваючою комою типу `float`;
- `%lf` – дійсне число із плаваючою комою типу `double`;
- `%o` – вісімкове ціле число без знака;
- `%s` – рядок символів;
- `%u` – десяткове число цілого типу без знака;
- `%x` – шістнадцяткове ціле число без знака (літерами нижнього регістру);
- `%X` – шістнадцяткове ціле число без знака (літерами верхнього регістру);
- `%p` – виводить на монітор значення вказівника;
- `%%` – виводить символ `%`;
- `%ld` – десяткове довге знакове ціле `long int`;
- `%lu` – десяткове довге беззнакове ціле `unsigned long int`;
- `%e` – виводить дійсні числа в експоненційному форматі.

Обернений слеш `\` у рядку екранує наступний символ. Щоб вивести на екран деякі символи, їх необхідно екранувати. Такі позначення називають ескейп-послідовностями. Часто використовують такі ескейп-послідовності:

- `\t` – табуляція;
- `\n` – новий рядок;
- `\r` – повернення на початок рядка;
- `\r` – повернення на символ назад;
- `\v` – вертикальний відступ;
- `\?` – виведення знака `?`;
- `\\` – виведення `\`;
- `\"` – виведення ”.

Приклад використання специфікаторів форматування та ескейп-послідовностей:

```
1  int a = 10, b = 20, c = 30;
2  printf("a = %d, b = %d, c = %d \n\v", a, b, c);
3
4  printf("c = %o (oct)\nc = %x (hex)\nc = %X (HEX)\n", c, c, c);
```

На екрані буде виведено:

```
a = 10, b = 20, c = 30


c = 36 (oct)
c = 1e (hex)
c = 1E (HEX)
```

При виведенні цілих чисел можна задавати кількість позицій на все число, а для дійсних – також на дробову частину. Розглянемо код:

```
1  printf("/%d/\n", 111);
2  printf("/%2d/\n", 111);
3  printf("/%10d/\n", 111);
4  printf("/%-10d/\n", 111);//
5  printf("/%12.8lf/\n", 3.14159);
```

Результатом його виконання буде:

```
/111/
/111/
/␣␣␣␣␣␣␣111/
/111␣␣␣␣␣␣␣/
/␣␣3.14159000/
```

У першому рядку бачимо звичайне виведення за умовчуванням. У другому рядку до специфікатора `%d` додано число 2, яке означає, що виділяється дві позиції на виведення 111. Задане ціле число тризначне, тому обмеження на кількість позицій буде проігнороване. Третя команда задає для виведення 10 символів. Як видно, буде виведено число 111, а решта позицій перед ним залишаться пустими. Почати виведення числа зліва можна, якщо поставити мінус перед кількістю символів, як у рядку 4. Для дійсного числа вказується повна кількість символів, а через крапку – скільки з них виділяється на дробову частину. Якщо кількість цифр у дробовій частині менша, ніж задана,

тоді зайві позиції будуть заповнені нулями (рядок 5).

Для введення інформації зі стандартного потоку вводу використовується функція `scanf`. Параметрами `scanf` є рядок формату і список аргументів. Суттєва відмінність у синтаксисі `scanf` і `printf` полягає в особливостях списку аргументів. Функція `printf` використовує імена змінних, констант та вирази, тоді як для `printf` вказують тільки список змінних, куди необхідно записати введені дані (точніше – список адрес цих змінних). Функція `scanf` використовує той самий символів специфікаторів форматування, що й функція `printf`. Наступна програма розрахує суму двох чисел, введених користувачем:

```
1  #include <stdio.h>
2  main()
3  {
4      int a, b, c;
5      printf("A = ");
6      scanf("%d", &a);
7      printf("B = ");
8      scanf("%d", &b);
9      c=a+b;
10     printf("A + B = %d", c);
11 }
```

Знак амперсанд (`&`) перед іменем змінної у `scanf` (рядки 6 і 8) обов'язковий у цій програмі і позначає операцію отримання адреси змінної. Якщо цей знак опустити, як правило, виникає помилка виконання програми "Segmentation fault (core dumped)".

### Основні оператори

Оператори застосовуються до виразів (операндів). Більшість операторів мають два операнди, один із яких розташовується перед знаком операції, а інший – після. За кількістю операндів розрізняють унарні, бінарні та тернарні оператори.

За типом операндів та результатом операції оператори в C поділяють на такі види: арифметичні, присвоєння, порівняння, логічні та побітові.

Арифметичні оператори використовують для виконання типових математичних дій. Список арифметичних операторів наведено у табл. 2.

Табл. 2: Арифметичні оператори

| Оператор | Дія | Приклад |
|---|---|---|
| + | Додавання | `x + y` |
| - | Віднімання | `x - y` |
| * | Множення | `x * y` |
| / | Ділення | `x / y` |
| % | Остача від ділення | `x % y` |
| ++ | Інкремент (збільшення числа на 1) | `++x, x++` |
| -- | Декремент (зменшення числа на 1) | `--x, x--` |

Оператори порівняння дають змогу порівняти два значення за певною умовою. Якщо умова виконується, то буде повернуто ціле число "1", у протилежному випадку – "0". Оператори порівняння мови C наведено у табл. 3.

Табл. 3: Оператори порівняння

| Оператор | Дія | Приклад |
|---|---|---|
| == | Тотожність | `x == y` |
| != | Не дорівнює | `x != y` |
| > | Більше ніж | `x > y` |
| < | Менше ніж | `x < y` |
| >= | Більне або дорівнює | `x >= y` |
| <= | Менше або дорівнює | `x <= y` |

Логічні оператори необхідні для обчислення логічних виразів між змінними або значеннями. У C є три логічні оператори, описані у табл. 4.

Табл. 4: Логічні оператори

| Оператор | Дія | Приклад |
|---|---|---|
| && | Логічне "І" | `x < 6 && x < 7` |
| \|\| | Логічне "АБО" | `x < 5 \|\| x > 8` |
| ! | Логічне "НІ" | `!(x < 2 && x < 5)` |

Оператори присвоєння використовують для надання змінній деякого значення. Основним серед операторів присвоєння є знак "=", а інші позначають скорочений запис арифметичних та логічних дій із присвоєнням результату (табл. 5). Варто пам'ятати, що в присвоєнні дії виконуються справа наліво.

Табл. 5: Оператори присвоєння

| Оператор | Приклад | Повний запис |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x - 5 |
| *= | x *= 5 | x = x * 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| &= | x &= 5 | x = x & 5 |
| \|= | x \|= 5 | x = x \| 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |

Порозрядні (побітові) операції застосовуються тільки до цілочислових операндів і працюють із їх двійковим представленням (табл. 6). Ці операції неможливо використовувати зі змінними дійсних типів, наприклад `double` або `float`.

Табл. 6: Порозрядні операції

| Оператор | Дія | Приклад |
|---|---|---|
| & | Побітове “І” | x & y |
| \| | Побітове “АБО” | x \| y |
| ~ | Побітове “НІ” | ~x |
| ^ | Виключне “АБО” | x ^ y |
| << | Побітовий зсув уліво | x << y |
| >> | Побітовий зсув управо | x >> y |

Оператор обчислення розміру `sizeof` повертає об’єм пам’яті, необхідний для розміщення в ній виразів або змінних вказаних типів:

```
int myInt;
double myDouble = 10.47;
char myChar;
printf("%lu\n", sizeof(myInt));
printf("%lu\n", sizeof(float));
printf("%lu\n", sizeof(myDouble));
printf("%lu\n", sizeof(myChar));
```

Оператор слідування “кома” пов’язує два довільні вирази. Список виразів, розділених комою, обчислюється зліва направо: Наприклад, код

```
a = 4;
b = a + 5;
```

можна записати так:

```
a = 4, b = b + 5;
```

У складних виразах послідовність виконання операцій визначається дужками, старшинством операцій, а при однаковому старшинстві – асоціативністю.

### Перетворення базових типів

Компілятор виконує автоматичне (неявне) перетворення типів даних, особливо в арифметичних виразах, коли найчастіше цілочислових тип перетворюється у тип із плаваючою комою. Значення типу `char` автоматично перетворюється в ціле. Якщо операнди мають різні типи, тоді перед виконанням операції молодший тип перетворюється до старшого. Результатом буде старший тип. Порядок автоматичного перетворення типів за старшинством такий:

```
char -> short int -> int -> unsigned int -> long ->
 unsigned -> long long -> float -> double -> long double
```

Явне зведення арифметичних типів у C виконується з метою конвертування значення одного типу даних у інший. Загальний синтаксис перетворення типу має два варіанти:

1. `(new_type)expression;`
2. `new_type(expression).`

Обидва варіанти виглядають так:

```
1 char l = 'D';
2 int i = int(l);//символ приводиться до цілого int
3 long lg = (long) l;//символ приводиться до цілого long int
```

При перетвореннях типів внаслідок заокруглення може втрачатися інформація. Наприклад, при виконанні коду

```
int a = 5, b = 12;
double d = b/a, e = 24, f = e/b;
printf("d = %lf, f = %lf\n", d, f);
```

буде виведено `d = 2.000000, f = 2.000000`. Тут значення `d` розраховано неправильно, тому що результат ділення двох цілих буде заокруглено до цілого, а потім відбудеться присвоєння результату. Виправити ситуацію дає змогу явне зведення `b` до `double` у виразі для обчислення `d`:

```
int a = 5, b = 12;
double d = (double)b/a, e = 24, f = e/b;
printf("d = %lf, f = %lf\n",d,f);
```

Результат: `d = 2.400000, f = 2.000000`.

### Бібліотека математичних функцій

Математичні функції описані у бібліотеці *math.h*. Для її використання на початку програми необхідно додати директиву підключення

```
#include <math.h>
```

Деякі функції бібліотеки *math.h* описано у табл. 7.

Табл. 7: Математичні функції

| Математична функція | Функція бібліотеки math.h | Опис |
|---|---|---|
| $\cos(x)$ | `cos(x)` | Обчислення косинуса |
| $\sin(x)$ | `sin(x)` | Обчислення синуса |
| $\mathrm{tg}(x)$ | `tg(x)` | Обчислення тангенса |
| $x^y$ | `pow(x,y)` | Обчислення x у степені y |
| $e^x$ | `exp(x)` | Обчислення експоненти числа |
| $\sqrt{x}$ | `sqrt(x)` | Обчислення квадратного кореня |
| $\ln(x)$ | `log(x)` | Обчислення натурального логарифму |
| $\lg(x)$ | `log10(x)` | Обчислення десяткового логарифму |
| $\lvert x\rvert$ | `abs(x)` | Обчислення абсолютного значення |

Приклад. Розрахувати значення виразу $f = \sqrt{sin(b^3+1)+1}$.

Код програми:

```c
#include <stdio.h>//необхідно для scanf і printf
#include <math.h>//необхідно для sqrt і pow
int main()
{
    double f = 0, b = 0;
    puts("Введіть значення b");
    scanf("%lf", &b);
    f = sqrt(sin(pow(b, 3) + 1) + 1);
    printf("Відповідь: f = %lf", f);
    return 0;
}
```

## Список рекомендованої літератури


- Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. С. 8-36.
- Шпак З.Я. Програмування мовою С. Львів: Оріяна-Нова, 2006. С. 19-64.
- Пінчук В.П., Лозовська Л.І. Програмування мовою С/С++ з прикладами та вправами : навчальний посібник. Запоріжжя, ЗНТУ, 2008. С. 14-52.


## Порядок виконання роботи

1. Опрацювати теоретичні відомості.
2. Написати програму, яка виводить на екран власне прізвище, ім'я та академічну групу.
3. Написати програму, яка розраховує значення арифметичних виразів із табл. 8 згідно з варіантом.
4. Написати програму згідно з варіантом завдання із табл. 9.
5. Оформити звіт до роботи.

## Запитання та завдання для самоперевірки

1. Які базові типи даних існують у С та чим вони відрізняються?
2. Описати класифікацію типів даних у мові програмування С.
3. Описати структуру програми мовою С.
4. Що таке специфікатор форматування?

Табл. 8

| **Варіант** | **Завдання** | |
|---|---|---|
| | 1 | 2 |
| 1,6,11 | $f = a + b^2 + c^3$ | $10 - \frac{b-4ac}{ab^2} + \frac{sin(a)}{cos(ac)}$ |
| 2,7,12 | $f = x_1x_2x_3 - x_2x_3x_1$ | $\frac{(x^2+1)-(x^2-1)}{sin(x)+y^2}$ |
| 3,8,13 | $f = (b^2 - 4ac)(1 - ac^2)$ | $\frac{1+a^4+5a}{ln(a)+7ad-d^9}$, $a > 0$ |
| 4,9,14 | $f = a^4 - b^3 + \frac{a+1}{b}$ | $\lg(b) + \frac{b^2-4ac}{ab^5+(a-d)}$, $b > 0$ |
| 5,10,15 | $f = xyz + \frac{x}{y-z}$ | $\frac{\|a-b\|}{ab^5+(a-d)}$ |

Табл. 9

| **Варіант** | **Завдання** |
|---|---|
| 1, 6, 11 | Задати значення часу в секундах у вигляді чотиризначного числа. Перевести цей час у години і хвилини та вивести на екран. |
| 2, 7, 12 | Задати значення часу в хвилинах за допомогою тризначного числа. Перевести цей час у секунди та вивести на екран. |
| 3, 8, 13 | Задати значення часу в мілісекундах за допомогою шестизначного числа. Перевести цей час у години і хвилини та вивести на екран. |
| 4, 9, 14 | Задати значення часу в хвилинах за допомогою двозначного числа. Перевести цей час у мілісекунди та вивести на екран. |
| 5, 10, 15 | Задати значення часу в секундах за допомогою шестизначного числа. Перевести цей час у години і хвилини та вивести на екран. |

5. Як здійснюється форматоване введення та виведення даних?
6. За якими правилами відбувається неявне перетворення типів даних?
7. Які логічні операції в С ви знаєте?
8. Написати таблиці істинності для логічних операцій.
9. Які порозрядні операції в С ви знаєте?
10. Що таке ескейп-послідовності і для чого їх використовують?
11. Запишіть таблицю істинності для виключного “АБО”.

12. Що отримаємо при виконанні побітового зсуву `7>>2`?

# Лабораторна робота 2
# Оператори галуження та циклу

**Мета роботи:** отримати навички із програмування галужених та циклічних алгоритмів.

## Теоретичні відомості

При реалізації складних алгоритмів виникають ситуації, коли подальші дії програми залежать від умов, що виникають у ході її виконання. У найпростішому випадку для цього досить конструкції "якщо ..., то ...", яка у мові С реалізується за допомогою операторів галуження (операторів умови). Оператори галуження надають програмісту засоби вибору наступних дій у залежності від результату перевірки умови.

### Оператор умови if ... else

Для реалізації повного або неповного галуження використовують оператор `if`. Загальний вигляд умовного оператора:

```
if (<логічний вираз>) {<серія команд 1>};
if (<логічний вираз>) {<серія команд 1>};
 else {<серія команд 2>};
```

Перша форма називається скороченою, друга – повною формою умовного оператора. Логічний вираз є умовою та записується у круглих дужках, а серії команд після операторів `if` та `else` – у фігурних дужках. Проте, якщо команда одна, дужки можна опустити.

У мові С відсутній булевий тип даних, тому будь-який ненульовий результат обчислення умови вважається істиною (True), а нуль – хибою (False). Для опису логічного виразу використовують логічні оператори та оператори порівняння, наведені у табл. 3 і 4.

При неповному галуженні спочатку розраховується логічний вираз, якщо його значення "1" (істина), тоді виконується серія команд 1, інакше – відбува-

ється вихід із оператора умови. При повному галуженні, якщо логічний вираз дорівнює “1”, виконується серія команд 1, в іншому випадку – серія команд 2. Наприклад, програма, яка розраховує значення функції

$$f(x) = \begin{cases} 1, & x > 0, \\ 0, & x \leq 0, \end{cases}$$

буде наступною

```c
#include <stdio.h>
int main()
{
    double f, x = 0;
    scanf("%lf", &x);
    if (x>0)//перевірка умови
        f = 1;//умова виконується
    else
        f = 0;//умова не виконується
    printf("f = %lf\n", f);//виведення результату
    return 0;
}
```

Серія команд після операторів `if` та `else` може включати вкладені оператори умови, отже можна реалізувати алгоритм галуження довільної складності.

### Тернарний оператор

*Тернарний оператор* є скороченим записом `if ... else` і використовується там, де умова та команди є простими виразами. Синтаксис тернарної операції такий:

`<логічний вираз>?<вираз_1>:<вираз_2>;`

Код програми з попереднього прикладу з використанням тернарного оператора можна переписати так:

```c
#include <stdio.h>
int main()
{
    double f, x = 0;
    scanf("%lf", &x);
    f = (x > 0) ? 1 : 0;
```

```
    printf("f = %lf\n", f);//виведення результату
    return 0;
}
```

### Оператор switch

Оператор `switch` уможливлює вибір програмою подальших дій, коли варіантів вибору більше, ніж два. Синтаксис `switch` такий:

```
switch (<вираз>)
{
case <константний-вираз_1>:<серія команд 1>; break;
case <константний-вираз_2>:<серія команд 2>; break;
.....................................................
case <константний-вираз_n>:<серія команд n>; break;
default:<серія команд>;
}
```

Після виконання виразу в заголовку оператора його результат послідовно порівнюється з константами-виразами, починаючи з верхнього, доки не буде встановлено їх відповідність. Далі виконується серія команд усередині відповідного блоку `case`, яка закінчується оператором `break`. Якщо жоден константний вираз не дорівнює значенню вхідного, тоді виконується серія команд за умовчанням після оператора `default`. Блоку `default` може не бути.

Команда `break` забороняє подальший перегляд константних виразів після виконання блоку `case`, перериваючи виконання `switch`. При відсутності `break` оператор `switch` буде виконуватися повністю.

Зазвичай оператор `switch` використовується тоді, коли програміст хоче, щоб була виконана тільки одна послідовність команд із декількох можливих.

Наступна програма обчислює задану арифметичну дію для двох цілих чисел:

```
#include <stdio.h>
int main()
{
    int a = 0, b = 0, r = 0;
    char c = 'a';
    puts("Введіть через пробіл два цілі числа та знак
    ↪ математичної дії (+,-,*,/):");
```

```
7       scanf("%d%d %c", &a, &b, &c);//пробіл між %d і %c має
    ↪   значення!
8       switch (c)
9       {
10          case '+': r = a + b; break;
11          case '-': r = a - b; break;
12          case '*': r = a * b; break;
13          case '/': r = a / b; break;
14          default:
15              puts("Помилка введення або невідома математична
    ↪   операція!");
16              return 0;
17      }
18      printf("%d %c %d = %d\n", a, b, c, r);
19      return 0;
20  }
```

### Оператори циклу

Цикли, або ітераційні структури, дають змогу багаторазово виконувати окремі команди чи серії команд. Кількість повторень може бути фіксованою або визначатися в процесі роботи на основі перевірок одної чи кількох умов.

У мові С є три циклічні оператори, які реалізують цикл із параметром, цикл із передумовою та цикл із післяумовою. Циклічні дії можна також запрограмувати з використанням рекурсії або оператора `goto`.

### Цикл while

Цикл із передумовою `while` має такий синтаксис:

`while (<умовний_вираз>)<серія команд>;`

Перевірка умови відбувається перед виконанням тіла циклу (серії команд). Якщо результат обчислення умови не дорівнює нулю, то виконується серія команд. Перед входженням до циклу `while` ініціалізують одну або декілька змінних для того, щоб умовний вираз мав якесь значення. У тілі циклу необхідно змінювати значення одної чи декількох змінних, що входять в умовний вираз, щоб у кінцевому випадку вираз перетворився в нуль і цикл завершився.

Потенціальною помилкою при програмуванні циклу `while`, як і циклу

будь-якого іншого типу, є запис такого умовного виразу, який ніколи не дорівнюватиме нулю. Такий цикл буде безкінечним, наприклад, код

```
while(x) printf("Infinite loop!\n");
```

для будь-якого x, відмінного від 0, буде виводити на екран монітора текст "Infinite loop!".

Наступна програма надрукує на екрані цілі числа від a до 0 в порядку зменшення:

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    while(a >= 0)
    {
        printf("a = %d\n", a);
        a--;
    }
    return 0;
}
```

Цикл завершить роботу, коли змінна a стане меншою за нуль.

### Цикл do .. while

У циклі з післяумовою do .. while перевірка умови відбувається після виконання тіла циклу. Синтаксис циклу такий:

do <серія команд>; while (<умовний_вираз>);

У цьому циклі серія команд виконується як мінімум один раз. Цикл завершує роботу, коли умовний вираз перетворюється в нуль (стає неістинним). Наступна програма, як і попередня, виводить числа від a до 0 в порядку зменшення, проте код написаний за допомогою циклу з післяумовою:

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    do
    {
```

```
8           printf("a = %d\n", a);
9           a--;
10      }
11      while(a >= 0);
12      return 0;
13  }
```

### Цикл із параметром for

Загальною формою циклу в C є цикл `for`. Конструкція `for` виглядає так:
`for (<вираз 1>; <вираз 2>; <вираз 3>) <серія команд>;`

Перший вираз виконується перед початком роботи `for`. Як правило, тут задають початкове значення параметра циклу, хоча й і інші дії не заборонені. Якщо параметр циклу задано раніше, вираз можна опустити, замінивши пустою командою ";".

У другому виразі описують умову виходу із циклу. Найчастіше умова полягає в порівнянні поточного значення параметра циклу з максимальним (мінімальним).

Третій вираз містить правило зміни параметра циклу з кожною наступною ітерацією.

Варто зазначити, що кожний із трьох виразів може бути відсутнім або використаний програмістом, як завгодно. Якщо другий і третій вирази відсутні, то для уникнення зациклення програмісту необхідно забезпечити вихід з циклу та зміну його параметра командами у тілі циклу.

Алгоритм роботи циклу з параметром формально можна описати так:

1. Якщо перший вираз наявний, то він вираховується.
2. Якщо другий вираз наявний, то він вираховується. Якщо отримано значення 0, то цикл припиняється, в іншому випадку цикл буде продовжений.
3. Виконується тіло циклу.
4. Вираховується третій вираз (якщо він є).
5. Відбувається перехід до п.2.

Управляти роботою всіх циклів можна операторами `break`, `continue` або `return`. Якщо в тілі циклу зустрічається команда `break`, то цикл достроко-

во завершується, і програма продовжує роботу з першої команди за межами циклу. Виконання в будь-якому місці тіла циклу оператора `continue` приводить до миттєвого переходу до кроку 4, тобто до завершення одного кроку циклу. Оператор `return` виконує вихід не лише з циклу, але й із функції або програми, у якій викликаний. Усередині циклічних операторів може зустрічатися, коли необхідно не тільки перервати цикл, але і повернути дані в програму(підпрограму) вищого рівня. Нижче наведено код програми, яка виведе десять цілих чисел, більших за задане, у порядку зростання, якщо вони не перевищують 100:

```c
#include <stdio.h>
int main()
{
    int a;
    int i;
    scanf("%d", &a);
    for(i = 1; i <= 10; i++)
    {
        if (a > 100) break;
        printf("a = %d\n", ++a);
    }
    printf("a = %d\n", a);
    return 0;
}
```

Всі оператори циклу в С взаємозамінні. Зручність і доцільність використання конкретного з них залежить від початкових умов та особливостей задачі, що розв'язується.

## Список рекомендованої літератури


- Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. С. 49-56.
- Шпак З.Я. Програмування мовою С. Львів: Оріяна-Нова, 2006. С. 85-109.


## Порядок виконання роботи

1. Опрацювати теоретичні відомості.

2. Розробити алгоритми розв'язання задач свого варіанта, записавши їх у вигляді блок-схем.
3. Написати програми згідно з варіантом завдань. Передбачити всі можливі ситуації в програмі так, щоб уникнути входження в безкінечний цикл. Забезпечити в програмі виведення коментарів з поясненням різних можливих ситуацій.
4. Оформити звіт до роботи.

## Варіанти завдань

### Варіант 1

1. Дано дійсні числа: $a$, $b$, $c$, $d$. Якщо $a < b < c < d$, то кожне число замінити найбільшим із них; якщо $a > b > c > d$, то числа залишити без змін; в іншому випадку всі числа замінити їх квадратами. Значення $a$, $b$, $c$, $d$ задати самостійно. Перевірити коректність роботи програми для трьох випадків: а) $a < b < c < d$; b) $a \geq b \geq c \geq d$; c) $a \leq b \leq c \leq d$. На екран вивести початкові та змінені значення $a$, $b$, $c$, $d$.

2. Написати програму для розрахунку загального опору паралельного та послідовного з'єднання із $n$ резисторів. Спосіб з'єднання, значення $n$ та опори резисторів вводити з клавіатури. На екран вивести значення розрахованого опору.

### Варіант 2

1. Дано дійсні додатні числа $a$, $b$, $c$, $d$. З'ясувати, чи можна прямокутник зі сторонами $a$, $b$ вмістити усередині прямокутника зі сторонами $c$, $d$, щоб кожна зі сторін одного прямокутника була паралельна чи перпендикулярна кожній стороні другого прямокутника. Значення $a$, $b$, $c$, $d$ задати самостійно. Перевірити коректність роботи програми для трьох випадків: а) $a < c,\ b > d$; b) $a > c,\ d < b$; c) $d > b > c,\ a < c$. На екран вивести значення $a$, $b$, $c$, $d$ та відповідь "так" або "ні".

2. Написати програму, яка обчислює границю послідовності

$$a_n = \frac{n}{\sqrt{n^2+1} + \sqrt{n^2-1}},$$

прийнявши її за таке значення $a_n$, коли $|a_{n+1} - a_n| < 10^{-3}$, $n = 1, 2, \ldots$. На екран вивести члени послідовності від $a_1$ до $a_n$ та номер члена $n$, для якого

справедлива задана вище нерівність.

### Варіант 3

1. Написати програму для переведення номіналу ємності конденсатора у скорочену форму ($pF$, $nF$, $\mu F$, $F$) за правилом:

$$C = \begin{cases} pF, \text{ якщо } C \leq 10^{-9} \text{ Ф}; \\ nF, \text{ якщо } 10^{-9} < C \leq 10^{-6} \text{ Ф}; \\ \mu F, \text{ якщо } 10^{-6} < C \leq 1 \text{ Ф}; \\ F, \text{ якщо } C > 1 \text{ Ф}. \end{cases}$$

На екран вивести округлене значення ємності з відповідним позначенням, наприклад, якщо ввели 0.00111, вивести 1110 $\mu$F.

2. Корінь рівняння знаходиться послідовними наближеннями за формулою

$$x_{n+1} = \frac{2 - x_n^3}{5}.$$

Написати програму, яка знаходить таке значення кореня, для якого різниця між двома сусідніми наближеннями не перевищує $10^{-3}$, виходячи з початкового значення $x_0 = 1$. На екран вивести знайдене значення кореня та порядок наближення $n$.

### Варіант 4

1. Написати програму для переведення номіналу опору резистора $R$ у скорочену форму ($\Omega$, $K\Omega$, $M\Omega$, $G\Omega$) за правилом:

$$R = \begin{cases} G\Omega, \text{ якщо } R \geq 10^9 \text{ Ом}; \\ M\Omega, \text{ якщо } 10^6 \leq R < 10^9 \text{ Ом}; \\ K\Omega, \text{ якщо } 1000 \leq R < 10^6 \text{ Ом}; \\ \Omega, \text{ якщо } 0 \leq R < 1000 \text{ Ом}. \end{cases}$$

На екран вивести округлене значення опору із відповідним позначенням, наприклад, якщо ввели 1100, вивести 1.1 K$\Omega$.

2. Обчислити кінцеву суму

$$\sum_{i=1}^{\infty} \frac{(-1)^{i+1}}{i(i+1)(i+2)}$$

із точністю $\varepsilon = 10^{-3}$, враховуючи, що необхідної точності досягнуто, якщо наступний доданок виявився за модулем менший, ніж $\varepsilon$. На екран вивести значення суми та номер доданка, на якому досягається задана точність.

**Варіант 5**

1. Задано три дійсних числа, що є довжинами відрізків. Визначити, чи можна побудувати трикутник із такими довжинами сторін і, якщо це можливо, визначити тип трикутника: рівносторонній, рівнобедрений, прямокутний чи інший трикутник. На екран вивести висновок про можливість побудови трикутника та його тип.

2. Написати програму для розрахунку загальної ємності паралельного та послідовного з'єднання $n$ конденсаторів. Спосіб з'єднання, значення $n$ та ємності конденсаторів вводити з клавіатури. На екран вивести розраховане значення ємності.

## Запитання та завдання для самоперевірки

1. Які оператори умовного переходу у мові C ви знаєте?
2. Поясніть роботу оператора `switch`.
3. Який синтаксис повного та скороченого оператора умови `if`?
4. Які оператори циклу ви знаєте?
5. Описати механізм роботи оператора циклу з передумовою.
6. Описати механізм роботи оператора циклу з післяумовою.
7. Описати механізм роботи оператора циклу з параметром.
8. Для чого використовуть оператори `break` та `continue`?
9. Що називають тернарним оператором?
10. Підготуватися за вимогою викладача внести в написаний код зміни для відображення процесу виконання програми.

# Лабораторна робота 3
# Робота з масивами

**Мета роботи:** отримати практичні навички роботи з масивами та вказівниками.

## Теоретичні відомості

**Масив** – це структурований тип даних, що складається з декількох елементів одного типу. Як тип масиву може бути використаний будь-який відомий (оголошений) тип мови С. Відмінність оголошення масиву від оголошення звичайної змінної полягає в наявності квадратних дужок після ідентифікатора масиву, в яких вказується кількість елементів у масиві. Приклад задання масиву:

```
int   a[10];//масив цілих чисел розміром 10 елементів
char  b[20];//масив символів розміром 20 елементів
float c[100];//масив дійсних чисел розміром 100 елементів
```

Таким способом створюють статичні масиви, розмір яких відомий на етапі компіляції і не може бути змінений під час роботи програми.

Масиви бувають одновимірні та багатовимірні. Розмірність масиву визначається під час оголошення за кількістю квадратних дужок після імені масиву. Наприклад:

```
int a[10];//одновимірний масив
int b[10][10];//двовиміний масив, 10*10=10 елементів
int c[5][5][10];//тривимірний масив, 5*5*10=250 елементів
```

Нумерація елементів масиву за кожним виміром починається з нульового елементу. Для масиву `int a[10];` максимальний індекс елемента 9.

Звернення до записаних у масиві даних виконується за іменем масиву та номером елемента. Наприклад, щоб присвоїти змінній `x` значення п'ятого елемента масиву `a`, необхідно написати код `x = a[4];`

Елементи масиву розміщуються в пам'яті послідовно. Якщо заданий опис масиву: `int a[2][3];` тоді елементи масиву `a` будуть розміщуватись у пам'яті таким чином: `a[0][0]`, `a[0][1]`, `a[0][2]`, `a[1][0]`, `a[1][1]`, `a[1][2]`.

Мова С дозволяє ініціалізувати масив під час оголошення. Для цього у фігурних дужках наводять список елеменів, розділених комою:

```
int  a[5]    = {0,1,2,3,4};
int  b[2][3] = {1,2,3,4,5,6};
char s[7]    = {'a','b','c','d','e','f','g'};
```

У випадку масиву `b` елементи набудуть значень: `b[0][0]=1`, `b[0][1]=2`, `b[0][2]=3`, `b[1][0]=4`, `b[1][1]=5`, `b[1][2]=6`.

### Опрацювання масивів

Зазвичай робота з масивом здійснюється за допомогою операторів циклу `for`, `while` або `do ... while`. Проілюструємо використання операторів циклу і функції `scanf` для введення значень масиву `int a[10];`:

```
...
//цикл for
for(i = 0; i < 10; i++)
{
    scanf("%d", &a[i]);
}
//цикл while
i = 0;
while(i < 10)
{
    scanf("%d", &a[i]);
    i = i + 1;
}
//цикл do while
i = 0;
do
{
    scanf("%d", &a[i]);
    i++;
}
while(i <= 10);
```

Для роботи з багатовимірними масивами використовують *вкладені цикли*: такі, де в межах одного циклічного оператора виконується інший циклічний оператор. Для двовимірного масиву спочатку ведеться перебір рядків, це зовнішній цикл. Потім елементи рядка (стовбці) переглядають у внутрішньому циклі. Наприклад, вивести на екран матрицю `a` розміром 3*3 можна так:

```c
int i, j, a[];
// ...
for (i = 0; i < 3; i++)
{
    for (j = 0; j < 3; j++)
        printf("%d\t", a[i][j]);
    printf("\n");
}
```

### Вказівники

Вказівник – це змінна, яка зберігає адресу деякого об'єкта (іншої змінної, константи, ...), через яку можна отримати доступ до цього об'єкта. Оголошують вказівник за допомогою оператора "*", який ставлять після типу та перед іменем вказівника:

```c
int *x;//вказівник на ціле число
char *c;//вказівник на символ
```

Після такого оголошення значення вказівника невизначене.

Ініціалізувати вказівник можна адресою раніше створеної змінної або спеціальним значенням NULL. Для отримання адреси призначений унарний оператор "&", який ставиться перед іменем змінної, адресу якої необхідно отримати. Оператор "&" можна використовувати тільки до змінних і елементів масиву. NULL означає, що за вказівником не закріплена адреса жодного об'єкта. Ініціалізація значенням NULL дає змогу перевіряти коректність вказівника у програмі. Якщо у тексті програми вказівник перевіряється на NULL, то після створення його необхідно прирівняти до NULL. Приклади оголошення вказівників із присвоєнням значення:

```c
int a = 10;
int *x = &a;//оголошено вказівник x типу int * та присвоєно
 ↪  значення адреси a
double d = NULL;//оголошено вказівник d типу double* та
 ↪  ініціалізовано в NULL
```

Щоб отримати значення за адресою у вказівнику, перед ним ставиться унарний оператор "*". Така операція називається розіменуванням вказівника.

```c
int a = 10, b;
int *x = &a;
b = *x;//змінній b присвоєно значення за адресою вказівника,
 ↪  після виконання цієї команди b=10
```

Значення вказівника при форматованому виведенні задають специфікатором %p, а у списку аргументів функції printf вказується адреса об'єкта:

```c
#include <stdio.h>
int main()
{
    char c = 'W', *pc = NULL;
    printf("Значення змінної c = '%c'\n", c);
    printf("Значення адреси змінної &c = '%p'\n", &c);
    printf("Значення вказівника pc = '%p'\n", pc);
    pc = &c;
    puts("Виконано команду \"pc = &c;\"");
    printf("Значення вказівника pc = '%p'\n", pc);
    printf("За адресою у вказівнику знаходиться pc = '%c'\n",
 ↪  *pc);
    printf("Адреса вказівника &pc = '%p'\n", &pc);
    return 0;
}
```

Результат виконання програми:

```
Значення змінної c = 'W'
Значення адреси змінної &c = '0x7ffda67ae98f'
Значення вказівника pc = '(nil)'
Виконано команду "pc = &c;"
Значення вказівника pc = '0x7ffda67ae98f'
За адресою у вказівнику знаходиться pc = 'W'
Адреса вказівника &pc = '0x7ffda67ae990'
```

Після присвоєння вказівнику pc адреси змінної c, значення вказівника та адреса змінної однакові. Причому вказівник як змінна має свій адрес, відмінний від його значення.

### Зв'язок масивів і вказівників

Ім'я масиву – це вказівник-константа, який містить адресу його першого елемента.

Над вказівниками дозволено ряд арифметичних операцій. Наприклад,

якщо y – вказівник на масив, то унарна операція y++ збільшує його значення до адреси наступного елемента. Вказівнику не можна присвоювати цілі значення, проте можна підсумовувати вказівники та цілі числа. Конструкція y+n (y – ім'я масиву, n – ціле число) задає адресу $n$-го об'єкта, на який вказує масив y. Це справедливо для будь-яких об'єктів (int, char, float і т.д.). Транслятор буде масштабувати зміну адреси відповідно до типу визначеного в оголошені (`int x[10]; char y[50]; float z[5];`). Якщо два вказівники посилаються на елементи одного і того ж масиву, то допускаються операції відношення над ними: ==, !=, <, >, і т.д. Для вказівників, що посилаються на елементи різних масивів, результат порівняння не визначений.

Ім'я масиву вказівників можна використовувати для роботи з його елементами без індексації у квадратних дужках. У програмі

```
#include <stdio.h>
int main()
{
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    int *b = a;//допустима операція, a - вказівник-константа
    printf("%d\n", *a); //так можна отримати тільки значення
    ↪ першого елемента масиву
    // a++; недопустима операція, a - вказівник-константа
    b++; printf("%d\n", *b);//b містить адресу другого
    ↪ елемента
    b += 5; printf("%d\n", *b);//b містить адресу сьомого
    ↪ елемента
    b += 15; printf("%d\n", *b);//вихід за межі масиву, спроба
    ↪ доступу до 22-го елемента
    return 0;
}
```

вказівник b ініціалізований масивом a. Зміною b та розіменуванням результату можна отримати доступ до об'єктів масиву. Присвоїти змінній x значення другого елемента масиву a можна двома способами: 1) `b++; int x = *b;` 2) `int x = a[1]`.

Працюючи з масивами через вказівники, програміст повинен самостійно убезпечити програму від виходу за межі виділеної пам'яті. Якщо обрано елемент, для якого не виділена пам'ять, то програма буде працювати неправильно, оскільки компілятор не стежить за цим. У ряді випадків відбудеться ава-

рійне завершення програми. Команда у рядку 10 може бути виконана, проте результат буде помилковий, оскільки при створенні розмір масиву обмежений 10-ма елементами.

## Список рекомендованої літератури


- Шпак З.Я. Програмування мовою С. Львів: Оріяна-Нова, 2006. С. 112-140.
- Пінчук В.П., Лозовська Л.І. Програмування мовою С/С++ з прикладами та вправами : навчальний посібник. Запоріжжя; ЗНТУ, 2008. С. 55-73.


## Порядок виконання роботи

1. Вивчити правила організації, синтаксис опису та принципи роботи з одновимірними та двовимірними масивами (матрицями).
2. Розробити алгоритми вирішення задач згідно з варіантом, записавши їх у вигляді блок-схем.
3. Написати програму з коментарями-поясненнями. Роботу з матрицями та векторами описати за допомогою індексації через "`[]`".
4. Виконати перші завдання згідно з варіантом, організувавши оброблення матриць та векторів за допомогою вказівників.
5. Оформити звіт до роботи.

### Варіанти завдань

#### Варіант 1

1. Помножити матрицю $A_{m\times m}$ на вектор $R$ з розміром $n$ за формулою

$$U_i = \sum_{j=1}^{n} a_{ij} * r_j,$$

де $i = 1, 2, ...m, j = 1, 2, ...n$. На екран вивести початкову матрицю $A$, вектор $R$ (задати самостійно) та результуючий вектор $U$.

2. Дано натуральне число $n$, цілочислова квадратна матриця $A$ порядку $n$. Отримати послідовність $b_1...b_n$, де $b_i$ – це найменше з значень елементів,

які знаходяться на початку $i$-го рядка матриці до елемента, який належить головній діагоналі включно. Матрицю $A$ задати самостійно. Вивести $A$ та $b_1...b_n$.

**Варіант 2**

1. Додати та відняти дві матриці цілих чисел $A_{m\times m}$ і $B_{m\times m}$ за формулами:

$$c_{ij} = a_{ij} + b_{ij};\ d_{ij} = a_{ij} + b_{ij};,$$

де $i = 1, 2, ...m, j = 1, 2, \ldots n$. На екран вивести початкові матриці (задавати самостійно) та результуючі.

2. Задана цілочислова квадратна матриця $A_{4\times4}$. Знайти найменше зі значень елементів стовпця, у якого найбільша за модулем сума елементів. Якщо таких стовпців декілька, то взяти перший із них. Початкову матрицю задати самостійно. Вивести початкову матрицю, номер стовпця та знайдений елемент.

**Варіант 3**

1. У заданій матриці дійсних чисел $A_{4\times4}$ поміняти місцями рядок, який містить елемент із найбільшим значенням, із рядком, який містить елемент із найменшим значенням. Припускається, що такі елементи єдині. Початкову матрицю задати самостійно. Вивести початкову і змінену матриці, номери обміняних місцями рядків.

2. Задана матриця дійсних чисел $A_{m\times n}$, де $|a_{i,j} < 10|$. Отримати послідовність $b_1...b_n$, де $b_i$ – це добуток квадратів тих елементів $i$-го рядка, модулі яких належать до діапазону $[1, 2]$. Початкову матрицю задати самостійно. Вивести початкову матрицю та послідовність $b_1...b_n$.

**Варіант 4**

1. У заданій матриці дійсних чисел $A_{m\times m}$ знайти суму елементів рядка, у якому знаходиться елемент із найменшим значенням. Припускається, що такий елемент єдиний. Початкову матрицю задати самостійно. Вивести початкову матрицю, найменший елемент та знайдену суму.

2. Задана послідовність цілих чисел $b_1...b_n$. Сформувати матрицю $A_{n\times n}$, у якій перший елемент $i$-го рядка це $b_i$, а кожен наступний на 1 менший від попереднього (наприклад, $b_i = 5$, $n = 4$, тоді у рядок матриці буде $5, 4, 3, 2$).

На екран вивести послідовність (задати самостійно) та створену матрицю.

**Варіант 5**

1. Задано двовимірний масив $A_{m\times n}$, елементами якого є цілі числа, що складаються з різної кількості цифр. Написати програму для вектора, елементами якого будуть цифри, з яких складається заданий елемент масиву $A$. Початкову матрицю задати самостійно, індекси заданого елемента вводить користувач за запитом програми. На екран вивести початкову матрицю та знайдений вектор.

2. Задане натуральне число $k$ та двовимірний масив цілих чисел $A_{k\times k}$. Поміняти у масиві максимальний елемент з першим, а мінімальний – із останнім. Значення $k$ та початковий масив задати самостійно. Вивести початковий та змінений масиви.

## Запитання та завдання для самоперевірки

1. Як задати одновимірний масив?
2. Поясніть, як задати багатовимірний масив.
3. Якого типу елементи можуть бути у масиві?
4. Як отримати доступ до елементів масиву?
5. Наведіть визначення вказівника.
6. Як пов'язані масиви і вказівники?
7. Які операції дозволені над вказівниками?
8. Як вивести значення вказівника?

# Лабораторна робота 4
# Програмування з використанням підпрограм користувача

**Мета роботи:** отримати навички з алгоритмізації і програмування задач із використанням підпрограм користувача.

## Теоретичні відомості

У мові С передбачено засоби, завдяки яким можна оформляти послідовність операторів як підпрограму. *Підпрограма* – група команд, об'єднана власним іменем, яку можна викликати та виконати в будь-якому місці програми довільну кількість разів.

Усі підпрограми поділяються на два класи: стандартні (зарезервовані) і визначені користувачем. Стандартні підпрограми є частиною мови, вони заздалегідь не описуються, проте для їх використання може знадобитися підключення бібліотек, які постачаються разом із компілятором. Підпрограми, визначені користувачем, обов'язково описуються. Сам опис є звичайним кодом і не передбачає виконання жодних дій.

Серед підпрограм розрізняють процедури та функції. Перші виконують певні дії без можливості присвоєння результату, другі – можуть повертати результат в основну програму (підпрограму). У мові С існують лише функції, але, якщо правильно описати підпрограму, то вона може працювати і як функція, і як процедура.

### Функції

Загальний синтаксис опису функції включає тип значення, яке повертається, ім'я, список формальних параметрів у круглих дужках та код у фігурних дужках:

```
<тип повернення> <ім'я функції> (<опис пар._1>,<опис пар._2>,...,
<опис пар._n>)
{<список команд>}
```

*Тип повернення* – це будь-який базовий або користувацький тип, відомий компілятору. Якщо функція не повертає значення, то вказується `void`. Інакше після оператора `return` необхідно задати вираз або значення, яке збігається з типом повернення. Виконання оператора `return` у довільному місці функції завершує її роботу, передає повернене значення у функцію, що викликала дану, і переходить до опрацювання команди, що йде після виклику функції.

Кількість параметрів функції може бути довільною.

Необхідно розрізняти заголовок та реалізацію функції. *Заголовком* (*сигнатурою*) функції називається її оголошення без розкриття основного коду у фігурних дужках. Заголовок закінчується ";":

`<тип повернення> <ім'я функції> (<опис пар._1>,<опис пар._2>,...,<опис пар._n>);`

Заголовок є декларацією для компілятора, що така функція існує та повідомляє її параметри.

Реалізація включає список команд для виконання. Заголовок та реалізація можуть бути розміщені в різних частинах або файлах коду програми. У простих навчальних програмах рекомендується заголовки функції писати до основної функції `main`, а їх реалізації – після.

Під час виклику функції вказуються фактичні параметри виклику, їх кількість та порядок мають збігатися з числом, типом і порядком формальних параметрів у заголовку цієї функції. Розглянемо програму

```
#include <stdio.h>
void print_my_name();//оголошення функції print_my_name
int summa_A_and_B(int, int);//оголошення функції з двома
 ↪  параметрами типу int
int main()
{
    print_my_name();
    int A = 1, B = 4;
    int i = summa_A_and_B(A, B);
    printf("A + B = %d\n", i);
    return 0;
}
void print_my_name()//опис функції
{
    puts("My name is Ivan!\n");
}
int summa_A_and_B(int A, int B)//опис функції
```

```
17 {
18     return A + B;
19 }
```

Функція `print_my_name`, яка не має параметрів і нічого не повертає, оголошена у рядку 2, викликана у рядку 6, а описана у рядках 12-15. Ця функція виводить рядок `My name is Ivan` на екран. Функція `summa_A_and_B`, оголошена у рядку 3, після виконання повертає цілочислове значення, яке присвоєно змінній `i` у рядку 8. Список формальних параметрів в оголошенні задано перерахуванням типів (імена змінних не вказано), проте в описі необхідно обов'язково вказати імена параметрів (рядок 16).

### Локальні і глобальні змінні

У мові С змінні діляться на глобальні і локальні. *Глобальні* змінні оголошуються у файлі початкового тексту програмного модулю поза будь-якою функцією, а *локальні* – всередині функції. Глобальні змінні створюються в точці оголошення, існують до закінчення роботи програми і доступні (видимі) у початковому тексті від точки оголошення до кінця файлу, в якому вони задані (в тому числі і в середині функцій, описаних у файлі). Глобальні змінні видимі також для зовнішніх модулів.

Локальні змінні по відношенню до функцій є внутрішніми. Життєвий цикл локальних змінних починається з оголошення (створення) в середині функції і закінчується після завершення її роботи. Якщо локальні змінні записані в списку параметрів функції (у круглих дужках), то треба розглядати таке оголошення як введення до першої відкриваючої фігурної дужки. Для тих локальних змінних, яких нема в списку параметрів, оголошення робиться у межах опису функції, обмеженого фігурними дужками.

У С заборонено визначати одну функцію в середині іншої, тому всі імена функцій є глобальними. Зв'язок між функціями здійснюється через аргументи, значення, що повертаються, і зовнішні (глобальні) змінні.

Локальні та глобальні змінні можуть мати однакові імена, але у деяких випадках це може створювати проблеми в роботі програми. Розглянемо код

```
1  #include <stdio.h>
2  int a = 10, b = 20; //глобальні змінні
3  int f1(int x); //x - локальний формальний параметр функції f1
4  int main ()
5  {
6      printf("a = %d\n", a);//a = 10
7      int c = 2;
8      int a = 45;
9      printf("a = %d\n", a);//a = 45
10     int d = f1(c);//c і d - локальні у main
11     printf("%d", d);
12     {
13         int a = 888;
14         printf("a = %d\n", a);//a = 888
15     }
16     printf("a = %d\n", a);//a = 45
17     return 0;
18 }
19 int f1(int x)
20 {
21     printf("a = %d\n", a);//a = 10
22     printf("b = %d\n", b);//b = 20
23     return (a + b) * x;
24 }
```

У рядку 2 `a` і `b` оголошені як глобальні змінні. Якщо вивести значення `a` (рядок 6), то отримаємо `a = 10`. У функції `main` оголошена та ініціалізована власна локальна змінна `int a = 45`, а її значеннння виводиться у рядку 9. При цьому значення глобальної змінної не видно `a`. Якщо викликати функцію `f1`, то для розрахунку буде використано глобальні змінні `a=10` і `b=20`.

Будь-який код, взятий у фігурні дужки, робить змінні (константи), створені в середині, локальними по відношенню до коду за межами цих дужок. Тому змінна `a`, створена у рядку 13, буде видалена після виконання рядків 14-15.

Якщо деякі змінні, константи, масиви або структури оголошені як глобальні, то їх можна не включати в список параметрів викликаної функції.

Часто виникає думка, що глобальні змінні – це універсальний спосіб обміну даними у програмі. Проте це не так. Глобальні дані можна змінювати у довільному місці програми, як наслідок, – ці зміни складно відстежувати і контролювати у великих проєктах. Варто також пам'ятати, що глобальні змінні видаляються лише при завершенні програми, а з метою захисту даних

та ефективного використання обчислювальних ресурсів не рекомендується їх зберігати у пам'яті зайвий час.

### Передавання даних у функції

У мові C параметри у функцію можна передавати за значенням або за адресою (вказівником). У першому способі відбувається копіювання даних у функцію. Наприклад функція із сигнатурою

```
int fun(int x, int y);
```

при виклику

```
int a = 2, b = 7, c;
c = fun(a, b);
```

отримує `x` і `y` за значенням. Локальним змінним буде присвоєно скопійовані `a` і `b` так, що стане `x = 2` і `y = 7`. Функція `fun` може як завгодно змінювати `x` і `y`, але не матиме доступу до `a` і `b`.

У разі використання другого способу параметри функції задаються як вказівники на змінні. Тоді при виклику передаватимуться адреси, за якими буде доступ до значень змінних. Самі значення не копіюються. Наприклад, функція з прототипом

```
int fun(int *x, int *y);
```

отримає адреси двох цілих чисел. Тоді при виконанні коду

```
int a = 2, b = 7, c;
c = fun(&a, &b);//передаємо адреси змінних a не їх значення
```

`fun` отримає адреси комірок пам'яті, де записано числа 2 і 7. Знаючи ці адреси, можна змінити значення `a` і `b`.

Якщо як аргумент функції використовується ім'я масиву, то передається початок масиву (адрес початку масиву), а самі елементи не копіюються. Функція може змінювати елементи масиву, рухаючись (індексуванням) від його початку до кінця.

У програмі нижче показано три варіанти сигнатур функцій, які отримують масив:

```c
#include <stdio.h>
void fun1(int a[10], int n);
void fun2(int a[], int n);
void fun3(int *a, int n);
int main()
{
    int b[10] = {0,1,2,3,4,5,6,7,8,9};
    fun1(b, 10);
    fun2(b, 10);
    fun3(b, 10);
    return 0;
}
void fun1(int a[10], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
void fun2(int a[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
void fun3(int *a, int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

Вказати масив як параметр можна по-різному: `a[10]`, `a[]`, `*a` (рядки 2-4), проте у всіх випадках передається розмір масиву `n`. Реалізація функцій `fun1`, `fun2` та `fun3` однакова і виконана через звернення за індексами.

Незалежно від вибраного варіанта, функції, що викликана, передається вказівник на початок масиву (рядки 8-10). Самі ж елементи масиву не копіюються.

## Список рекомендованої літератури


- Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. С. 61-73.
- Шпак З.Я. Програмування мовою С. Львів. Оріяна-Нова, 2006. С. 183-

232.

- Пінчук В.П., Лозовська Л.І. Програмування мовою C/C++ з прикладами та вправами : навчальний посібник. Запоріжжя, ЗНТУ, 2008. С. 76-86.

## Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Вивчити механізми передачі параметрів до функції.
3. Розробити програми, що розв'язують задачі згідно з варіантом.
4. Оформити звіт із поясненнями до коду програм.

### Варіанти завдань

### Варіант 1

1. Написати функції, які обчислюють залежності

$$x = \frac{a^b + b^a}{a^c + a^d};\ y = (x+10)^5 - (x-15)^2;\ z = x^{\sin a} + (x-10)^{\cos b};$$

Значення $a$, $b$, $c$, $d$ задати самостійно. Операцію піднесення до степеня оформити у вигляді функції. Функція для розрахунку `z` повинна повертати значення типу `void`.

2. Дано дійсні додатні числа: $a$, $b$, $c$. Написати функцію для розрахунку виразу

$$x = \frac{max(c, a+b) + max(a, a+b)}{5 - max(a+bc, b^2, 2^b)};$$

Значення $a$ і $b$ задати самостійно. Визначення максимального значення `max` оформити у вигляді функції, яка повертає тип `void`.

### Варіант 2

1. Задані дійсні числа: $a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$. Розрахувати величину

$$K = \begin{cases} min(a_1, a_2, a_3) + min(b_1, b_2, b_3), \text{ якщо } |min(a_1, b_1, c_1) > 5|, \\ 1 + (max(\sum_{i=1}^{3} a_i, \sum_{i=1}^{3} c_i))^3, \text{інакше.} \end{cases}$$

Значення $a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$ задавати самостійно. Визначення

`max`, `min` та `sum` ($\sum$) оформити у вигляді функції, які нічого явно не повертають.

2. Написати функції для обчислення опору паралельного та послідовного з'єднання з $n$ резисторів. Як параметри використати кількість резисторів і масив із номіналів опорів цих резисторів. За допомогою створених функцій порахувати опір для паралельного та послідовного з'єднання таких резисторів з опорами в Омах: *а)* 100, 470, 630, 6236; *б)* 11, 111, 18110, 4653, 25; *в)* 56, 456334, 234546.

**Варіант 3**

1. Написати функції для переведення числа із двійкової до десяткової, вісімкової та шістнадцяткової систем числення. Значення числа в заданій формі повертати як масив символів та виводити за допомогою окремої функції на екран.

2. Дано дійсні числа: $a_1, b_1, a_2, b_2, ..., a_n, b_n$. Написати функцію для розрахунку периметра многокутника, вершини якого мають відповідні координати: $(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)$. Операцію обчислення відстані між двома точками, які задані своїми координатами, оформити у вигляді аналога процедури. Числа $a_1, b_1, a_2, b_2, ..., a_n, b_n$ задавати самостійно.

**Варіант 4**

1. Написати функції для обчислення ємності паралельного та послідовного з'єднання $n$ конденсаторів. Як параметри підпрограм використати кількість конденсаторів $n$ і масив із номіналів цих конденсаторів.

За допомогою створених підпрограм порахувати ємність для паралельного та послідовного з'єднання таких конденсаторів ємністю (в пікофарадах): *а)* 100, 470, 630, 6235; *б)* 11, 111, 10110, 4653, 25; *в)* 56, 456324, 234546.

2. Написати функцію для обчислення виразу:

$$a = \begin{cases} 5 * x^2 + \sum_{i=0}^{k}(i^2 + \frac{x}{i+1}), \text{ якщо } 0 < x \leq 8 \\ 4 * x^3 - \sum_{i=0}^{k}(i + \frac{x}{i-1}), \text{ якщо } 8 < x < 12, \\ 0, \text{інакше.} \end{cases} .$$

Розрахунок суми оформити у вигляді процедури. Значення $x$ і $k$ вводити з клавіатури ($5 < k < 20$).

### Варіант 5

1. Задано натуральні числа $a$ і $b$. Написати функції для розрахунку: $x = max(a, b),\ y = min(a/b, a - b),\ z = max(x^2 + y^2, 100)$.

Значення $a$ і $b$ задавати самостійно з клавіатури. Обчислення `max` оформити у вигляді функції з типом повернення `void`.

2. Дано натуральні числа $x$ і $y$. Знайти значення суми $z(1, a) + z(x, y) - z(x^2, y^2)$, де

$$z(x, y) = \frac{x^2 + y^2 + 10}{x^2 + 2xy + y^5 - 1}.$$

Значення $x$ і $y$ задавати самостійно з клавіатури. Знаходження $z(x, y)$ та піднесення до степеня оформити у вигляді функцій із типом повернення `void`.

## Запитання та завдання для самоперевірки

1. Що таке функція в програмуванні?
2. Як ви розумієте різницю між формальними та фактичними параметрами функції?
3. Які способи передавання даних у функцію ви знаєте?
4. Як передаються парамети до функції і повертаються значення?
5. Опишіть різницю між локальними та глобальними змінними.

# Лабораторна робота 5
# Символьні дані

**Мета роботи:** отримати навички з алгоритмізації та програмування задач із оброблення символьних даних; засвоїти функції введення, виведення та оброблення символьних даних.

## Теоретичні відомості

У мові C масив символів який закінчується `'\0'`, називається *рядком*. Оголошення та ініціалізація рядків можлива кількома способами:

```
1 char s1[]= "Hello, world!";
2 char s2[] = {'H','e','l','l','o',',','\',
  ↪  ','w','o','r','l','d','!','\0'};
3 char *s3 = "Hello world!";
```

Рядки `s1`, `s2` та `s3` міститимуть однакову фразу `Hello, world!`. У першому і третьому випадках додавання символу закінчення рядка `'\0'` виконує компілятор, у другому – це необхідно зробити самостійно. У всіх наведених випадках розміри рядків визначаються автоматично.

Кількість символів у рядку (із врахуванням нульового) можна вказати у квадратних дужках:

```
1 char s1[21] = "Hello, world!";
```

У цьому випадку максимальний розмір рядка обмежено 21 символом, але зайнятими будуть перші 14, причому 14-й – це `'\0'`. У комірках пам'яті розміщення символів буде таким

| H | e | l | l | o | , |  | w | o | r | l | d | ! | \0 |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Після зміни рядка на

```
s1="Greetings!"
```

у пам'яті буде

| G | r | e | e | t | i | n | g | s | ! | \0 | ! | \0 |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

При обробленні масиву кінець рядка визначається перевіркою значення символу. Якщо знайдено '\0', тоді рядок закінчився.

Для виведення рядків призначений специфікатор %s:

```
printf("%s\n", s1);
```

У загальному випадку при форматованому виведенні рядків функції printf передається значення рядка, тобто адреса його першого символу. Якщо перший символ нульовий, то робота функції printf закінчується, а якщо ні, то вона виводить його на екран, додає до адреси одиницю і знову починає перевірку на нульовий символ. Така обробка дозволяє зняти обмеження на довжину рядка (звичайно, в межах оголошеної розмірності): рядок може бути будь-якої довжини доти, поки є місце в пам'яті, куди його можна помістити.

Рядки можуть бути елементами масиву. Двовимірний масив вказівників можна ініціалізувати, тобто назначати його елементам конкретні адреси рядків під час оголошення. Якщо задати масив

```
char s4[2][10]
```

і ініціалізувати його даними "Good Morning!", то буде виділено блок пам'яті в 20 байт, заповнений так:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| G | o | o | d | \0 | | | | | |
| m | o | r | n | i | n | g | ! | \0 | |

У порожніх комірках можуть бути довільні дані, що зберігалися раніше.

Оброблення елементів такого масиву можливо посимвольне або порядково:

```
#include <stdio.h>
int main()
{
    int i;
    char str_array[2][10] = {"Good","morning!"};
    for (i = 0; i < sizeof(str_array); i++)
        printf("%c:", *(str_array[0] + i));//посимвольне
 ↪  виведення
    printf("\n");
    printf("%s ", str_array[0]);//порядкове виведення
```

```
10      printf("%s", str_array[1]);
11      return 0;
12  }
```

### Операції над рядками

Засоби для роботи з рядками зібрані у бібліотеці заголовків *string.h*. Нижче наведено перелік та призначення функцій, які часто використовуються:

- `char *strcat(char *restrict dest, const char *restrict src);` – об'єднання рядків. Функція додає до рядка `dest` рядок `src` і повертає копію `dest`.
- `int strcmp(const char *lhs, const char *rhs);` – порівняння рядків. Функція порівнює рядки і повертає від'ємне значення, якщо `lhs < rhs`, нульове, якщо `lhs = rhs`, і додатне, якщо `lhs > rhs`. Порівняння виконується попарно до перших неоднакових символів або до закінчення одного чи обох рядків. Порядок визначається за номерами символів у таблиці ASCII.
- `char *strcpy(char *restrict dest, const char *restrict src);` – копіювання рядків. Функція копіює рядок `src` до кінця рядка `dest` і повертає `dest`. У `dest` має бути достатньо місця для запису `src`.
- `size_t strlen(const char *str);` – повертає довжину рядка (не враховуючи символ завершення `\0`).
- `char *strstr(const char* str, const char* substr);` – шукає підрядок `substr` у рядку `str`. У разі успішного пошуку функція повертає вказівник на елемент у `str`, з якого починається `substr`, інакше повертає `NULL`.
- `char *strchr(const char *str, int ch);` – пошук символу `ch` в рядку `str`. Повертає вказівник на перший символ, який збігається з `ch`, інакше повертає `NULL`.

Ключове слово `restrict` означає, що за оголошеним вказівником закріплена область пам'яті, з якою не пов'язаний жоден інший вказівник. Забезпечення унікальності вказівника гарантує програміст.

### Конвертація у числові формати

Функції для роботи з символьними даними з бібліотеки *stdlib.h* переводять рядки у числа:

- `double atof(const char* str);` – функція перетворює рядок `str` у число типу `double`.
- `int atoi(const char *str);` – функція перетворює рядок `str` у число типу `int`.
- `long atol(const char *str);` – функція перетворює рядок `str` у число типу `long`.
- `double strtod(const char *str, char **str_end);` – функція перетворює рядок `str` у числове представлення до символу, на який вказує `str_end`.

Перераховані функції приймають рядки, що містять послідовності цифр та знаки `'+'`, `'-'`, `'.'`. Інші символи ігноруються.

Приклад використання функцій роботи з рядками та конвертації наведено нижче.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  int main()
5  {
6      char s1[30] = "My name is ";
7      char s2[] = "Kate";
8      strcat(s1, s2);//об'єднання рядків s1 і s2
9      printf("%s\n", s1);
10     printf("%s\n", strchr(s1,'K'));//буде виведено частину
   ↪   рядка, починаючи з символу 'K'
11     char s3[] = "987665";
12     int a = atoi(s3);//перетворення рядка s3 у ціле число
13     printf("%d\n", a);
14     char *end;
15     printf("%lf\n", strtod("   0.0000321tutu",
   ↪   &end));//перетворення рядка у дійсне число
16     printf("%s\n", end);//виведення частини рядка починаючи з
   ↪   end
17     return 0;
18 }
```

У рядках 6 і 7 задано й ініціалізовано дві рядкові змінні `s1` і `s2`. Після до-

давання `s2` до `s1` значенням `s1` буде “My name is Kate”. Пошук символу “K” у `s1` функцією `strchr` поверне посилання на “K”. Функція `printf` (рядок 10) виведе решта рядка, починаючи з “K”. Конвертація до дійсного числа рядка “ `0.0000321tutu`” поверне `0.000032`, ”`tutu`” будуть проігноровані (рядок 15). Вказівник на вказівник на символ `end` збереже адресу відкинутого підрядка.

### Перевірка та перетворення символів

У бібліотеці *ctype.h* описані функції для класифікації та перетворення символів. Приклади таких функцій:

- `isalnum` – перевіряє, чи символ цифра або буква;
- `isalpha` – перевіряє, чи символ буква;
- `islower` – перевіряє, чи символ нижнього регістру;
- `isupper` – перевіряє, чи символ верхнього регістру;
- `isdigit` – перевіряє, чи символ цифра;
- `isspace` – перевіряє, чи символ пробіл;
- `ispunct` – перевіряє, чи символ знак пунктуації;
- `tolower` – перетворює символ до нижнього регістру;
- `toupper` – перетворює символ до верхнього регістру.

Параметром усіх перерахованих методів є один символ таблиці ASCII. За винятком двох останніх, наведені методи повертають ненульове значення, якщо умова виконується, і нуль – у протилежному випадку:

```
1  #include <stdio.h>
2  #include <ctype.h>
3  int main()
4  {
5      printf("isalnum('R') %s\n", isalnum('R') != 0 ? "True" :
   ↪  "False");
6      printf("isdigit('R') %s\n", isdigit('R') != 0 ? "True" :
   ↪  "False");
7      printf("isupper('R') %s\n", isupper('R') != 0 ? "True" :
   ↪  "False");
8      printf("tolower('R') %c\n", tolower('R'));
9      return 0;
10 }
```

Результат виконання програми:

```
isalnum('R') True
isdigit('R') False
isupper('R') True
tolower('R') r
```

Засоби роботи з рядками та символами використовують для перетворення та парсингу даних.

## Список рекомендованої літератури

- Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. С. 103-116.
- Шпак З.Я. Програмування мовою С. Львів : Оріяна-Нова, 2006. С. 141-163.

## Порядок виконання роботи

1. Вивчити правила створення та оброблення символьних даних (констант, змінних, масивів, рядків);
2. Ознайомитися з основними стандартними функціями, які використовуються для роботи з рядками. Звернути увагу на виклик підпрограм, типи формальних параметрів та повернене значення.
3. Написати програму згідно з варіантом. Забезпечити в програмі виведення коментарів та пояснень різних можливих ситуацій.
4. Оформити звіт до лабораторної роботи.

### Варіанти завдань

### Варіант 1

1. У кіоску продається газета, яка коштує 35 гривень, та журнал, який коштує 220 гривень. Написати програму, яка запитує про бажання покупця (журнал чи газета), приймає гроші (сума грошей вводиться з клавіатури) та виводить на екран здачу.

2. Написати програму, яка генерує випадкову послідовність символів латинського алфавіту та виводить її в оберненому порядку за рекурсивним

алгоритмом. У програмі має бути дві функції: одна для генерування послідовності, друга – рекурсивна, для посимвольного виведення послідовності на екран.

3. Рядок містить інформацію про особу: прізвище, ім‘я, по батькові та рік народження, розділені пробілом. Задано масив таких рядків. Визначити та вивести на екран середній вік осіб, прізвище найстаршої, ім'я найменшої.

**Варіант 2**

1. У заданому рядку знайти найкоротше і найдовше слово та вказати номери позицій, із яких вони починаються. Початковий рядок задавати самостійно.

2. Заданий довгий рядок, який містить тільки символи латинського алфавіту. Порахувати та вивести на екран в алфавітному порядку список символів заданого рядка у форматі "символ – скільки разів зустрічається". Видалити з рядка символи, яких найбільше, вивести результат. Алгоритми сортування не використовувати.

3. Задати інформацію про результати сесії для групи студентів у вигляді масиву рядків, кожний елемент якого містить інформацію про одного студента (прізвище та п'ять оцінок через кому). Обчислити та вивести на екран середній бал кожного студента.

**Варіант 3**

1. У масиві із $n$ рядків, кожен із яких містить ім‘я та прізвище, знайти тих, кого звуть Ростислав. Початковий масив задати самостійно.

2. Написати програму, яка приймає вказівник на рядок, рахує у ньому кількість букв і кількість цифр та виводить їх на екран. Підрахунок символів оформити як функцію.

3. Задано масив рядків, кожен з яких містить інформацію про клієнтів магазину у форматі "Прізвище, загальна сума покупок, кількість чеків", наприклад, "Козак, 10432, 4". Знайти та вивести середнє значення ціни в чеку та прізвище клієнта із найбільшою середньою ціною в чеку.

**Варіант 4**

1. Задано двовимірний масив символів, що містить список рядків – прізвищ деяких осіб. Написати програму, яка запитує ім'я, порівнює з тими, що вона має як елементи масиву, та виводить привітання або повідомляє, що

особа “незнайома”.

2. Дані у вигляді місячної заробітної плати робочих зі спеціальностями різних категорій занесені в одновимірний масив рядків, кожен із яких (рядок) містить: прізвище, категорію, місячний заробіток, номер цеху розділені символом “#”. Обчислити та вивести загальну суму виплат за місяць по всіх категоріях разом та по кожній окремо.

3. Задано масив рядків, таких, що кожен містить одне слово. Користувач вводить шукане слово. Завданням програми є пошук та виведення на екран слова (слів), які найкраще співпадають із заданим за кількістю символів, починаючи з першого. Наприклад, якщо у масиві записано слова “зачервоніло, коза, червоний”, а задане слово “червоніє”, то відповідь програми – слово “червоний”.

**Варіант 5**

1. У масиві рядків, кожен з яких містить прізвище, ім’я та по-батькові розділених пробілами, знайти й вивести на екран тих, у кого прізвище Гайдичук чи Савчук. Початковий масив задати самостійно так, щоб показати коректну роботу програми.

2. У символьному рядку видалити всі непарні числа, а парні числа взяти в круглі дужки (наприклад: початковий рядок ‘Abc18Cd056k-!B,10’, результуючий рядок :‘Abc(8)Cd(0)(6)k-!B,(0)’).

3. У рядку записані слова розділені пробілами. Програма має перетворити цей рядок таким чином, щоб букви всіх слів стали в оберненому порядку без зміни порядку слів. Для запису слова з кінця до початку написати та використати рекурсивну функцію.

## Запитання та завдання для самоперевірки

1. Що таке рядок у мові С?
2. Якими способами можна задати рядок у С?
3. Який специфікатор призначений для форматованого виведення рядків?
4. У чому відмінність рядка від масиву чисел?
5. Які функції для роботи з рядками ви знаєте? У якій бібліотеці вони описані?

6. Які функції для конвертування рядків у числа ви знаєте?

# Лабораторна робота 6
# Робота зі структурами та файлами

**Мета роботи:** отримати навички з алгоритмізації та програмування задач із використанням файлових структур даних.

## Теоретичні відомості

*Структура* – це об'єднання одного чи більше об'єктів (змінних, масивів, вказівників, інших структур і т.д.) під одним типом даних. Як і масив, структура представляє собою сукупність даних. Відмінністю є те, що до її елементів необхідно звертатися за ім'ям і те, що різні елементи структури можуть мати різний тип. Структура дає змогу програмісту задати власний користувацький тип даних.

Оголошення структури здійснюється за допомогою ключового слова `struct`, за яким іде назва і список елементів, що взяті у фігурні дужки:

```
struct <ім'я типу структури> {
<тип поля 1> <ім'я поля>,... ;
.................................
<тип поля n> <ім'я поля>,...;};
```

Ім'я типу структури – це ім'я типу даних, яке вказують при створенні об'єкта. Типом поля може бути будь-який відомий базовий або користувацький тип даних. Оголошення структури обов'язково завершується знаком ";". Наприклад, код

```
1  struct date
2  {
3      int day;
4      int month;
5      int year;
6  };
```

оголошує структуру `date`, яка об'єднує три цілі числа. Такий опис структури без наступного списку змінних не вимагає виділення ніякої пам'яті, а просто задає оголошення нового типу даних.

Створити об'єкт (змінну) з типом структури можна, задавши його ідентифікатор одразу після закривної дужки }:

```
struct date
{
    int day;
    int month;
    int year;
} my_date;
```

де my_date – змінна типу date. У цьому випадку кожній змінній виділяється пам'ять, об'єм якої визначається сумою розмірів всіх елементів структури. Змінна my_date містить три цілі числа day, month та year розміром 4 байт кожне, тому займатиме в пам'яті 4*3=12 байтів.

Ім'я типу date можна використовувати для оголошення змінної, але необхідно додатково вказувати ключове слово struct:

```
struct date new_date;
```

Тепер змінна new_date має тип date.

Ініціалізувати змінну типу структури можна під час створення або при зверненні до кожного її елементу в іншому місці коду. Наприклад, при оголошенні new_date присвоїти початкові значення можна так:

```
struct date new_date = {16, 10, 1999};
```

або

```
struct date new_date = {.day=16, .month=10, .year=1999};
```

Ще один спосіб використовує приведення до типу struct date виразу зі списком значень у фігурних дужках:

```
new_date = (struct date) {16, 10, 1999};
```

Порядок та типи значень у {.} та в описі структури повинні збігатися.

Щоб звернутися до окремого елементу, вказують розділені крапкою ім'я структури та ім'я потрібного елемента, наприклад:

```
my_date.day = 16;//змінній day структури my_date присвоєно
 ↪  значення 16
my_date.month = 10;
my_date.year =1999;
```

```
int d = my_date.day;//змінній d присвоєно значення змінної day
 ↪  структури my_date
```

Дозволено структури, вкладені одна в іншу, наприклад:

```
struct life
{
    sturct date birthday;
    sturct date deathday;
} granddad_life;
```

Структура `life` складається з двох змінних типу `date`. Для звернення до полів вкладених об'єктів треба вказувати повний шлях:

```
granddad_life.birthday.day = 19;
```

Можна створювати масиви структур:

```
struct life mayors[20];
```

Тут оголошений масив `mayors`, що складається із 20 структур типу `life`. Змінні структур одного типу можна присвоювати одна одній:

```
struct date new_date = {16, 10, 1999};
sturct date birthday;
birthday = new_date;//присвоєння birthday значення new_date
```

При цьому відбувається побітове копіювання даних з однієї змінної в іншу.

У процесі роботи із вказівниками на структури для отримання значень полів використовують операцію розіменування:

```
struct date new_date = {16, 10, 1999};
sturct date *p_new_date = &new_date;//оголошено вказівник
 ↪  p_new_date, який ініціалізовано адресою new_date
printf("%d\n", (*p_new_date).day);//виведення значення day
```

Круглі дужки в `(*p_new_date).day` необхідні, оскільки крапка має більш високий пріоритет, ніж розіменування.

Щоб уникнути розіменування, із вказівниками на структури можна використовувати оператор `->`, який спрощує представлення розглянутих вище конструкцій. Вираз `a = p_new_date->day` аналогічний до `a = (*p_new_date).day`.

Приклад. Написати програму, яка знаходить відстань між точками у три-

вимірному просторі.

Код програми:

```
#include <stdio.h>
#include <math.h>
struct tpoint //опис структури tpoint
{
    double x;
    double y;
    double z;
};
int main()
{
    double distance = 0;
    struct tpoint point1, point2; //створюється дві точки
    ↪ point1 і point2
    point1.x = 1;  //в point1 записується координата (1,1,1)
    point1.y = 1;
    point1.z = 1;

    point2.x = 5; //в point2 записується координата (5,5,5)
    point2.y = 5;
    point2.z = 5;
    //рахується відстань distance:
    distance = sqrt(pow(point2.x-point1.x, 2)+
        pow(point2.y-point1.y, 2)+pow(point2.z-point1.z, 2));
    printf("%10.5lf\n", distance);//виведення distance
    return 0;
}
```

У програмі координати точки об'єднані у структурі `tpoint`. Таке представлення підвищує рівень абстракції подання даних.

*Об'єднання* – це різновид структури, котра може зберігати в один момент часу тільки один об'єкт із кількох, різного типу і розміру. У результаті з'являється можливість зберігати в одній і тій же ділянці пам'яті дані різного роду. Для опису об'єднання використовується ключове слово `union`, а відповідний синтаксис аналогічний синтаксису структури:

```
union person
{
    long int id;
    struct date birthdate;
    char adress[150];
} person_1;
```

Тут `id` має розмір 8 байтів, `birthdate` – 12 байтів і `adress` – 150 байтів. Для `person_1` буде виділена пам'ять, достатня, щоб зберегти найбільшу змінну із трьох наведених. Отже, `person_1` використає 150 байтів. В один і той же момент часу коректне значення у `person_1` може мати тільки одна із вказаних змінних `id`, `birthdate` або `adress`.

Дослідити об'єднання `person` можна за допомогою такої програми:

```
1  #include <stdio.h>
2  struct date
3  {
4      int year;
5      int month;
6      int day;
7  };
8  union person
9  {
10     long int id;
11     struct date birthdate;
12     char adress[150];
13 };
14 int main()
15 {
16     union person student;//створення об'єднання student
17     student.birthdate.day = 5;
18     student.birthdate.month = 8;
19     student.birthdate.year = 1999;
20
21     printf("birthDate: %d %d %d\n", student.birthdate.year,
22     student.birthdate.month, student.birthdate.day);
23     printf("Adress: %s\n", student.adress);
24     student.id = 3176789;
25     printf("Id: %ld\n", student.id);
26     printf("birthDate: %d %d %d\n", student.birthdate.year,
27     student.birthdate.month, student.birthdate.day);
28     return 0;
29 }
```

```
birthDate: 1999 8 5
Adress:
Id: 3176789
birthDate: 3176789 0 5
```

Після створення об'єднання `student` (рядок 16) проведено ініціалізацію структури `birthdate`. Значення змінних birthdate виведено у рядку 21. При спробі

отримати адресу на екран не видно нічого – символи рядка `adress` неможливо зобразити на екрані. Об'єднання `student` після запису `id` (рядок 24) зберігає тільки це значення, а `birthdate` буде зміненим (рядок 26).

### Файли

*Файл* – це набір даних, що розташований на зовнішньому носії. У файлі розміщуються дані, що призначені для тривалого зберігання. Кожному файлу надається унікальне ім'я.

У мові С відсутні окремі засоби для роботи з файлами. Всі необхідні дії виконуються через функції, що включені в стандартну бібліотеку. Вони дозволяють працювати з різними пристроями введення та виведення інформації, такими, як: жорсткі диски, флешки, принтер, екран, клавіатура, комунікаційні порти і т.д. Ці пристрої дуже відрізняються один від одного. Однак файлова система дозволяє їх об'єднувати в один абстрактний логічний пристрій, що називається потоком. Існує два типи потоків: текстові і двійкові. Різниця між ними – в наборі символів таблиці ASCII, які можуть бути використані: текстові потоки оперують символами, які можна вивести на друк, двійкові – довільними.

Алгоритм роботи з файлами:

1. Відкрити (створити) файл.
2. Виконати необхідні дії (читання, запис, зміна, пошук інформації).
3. Закрити файл.

Засоби для роботи з файлами описані у бібліотеці `stdio.h`.

Відкривають файли за допомогою бібліотечної функції `fopen` із сигнатурою

```
FILE *fopen(const char *filename, const char *mode);
```

де `FILE` – ім'я типу, яке позначає потік в С; `filename` – ім'я файлу як рядкова константа; `mode` – рядкова константа, яка позначає тип файлу. Якщо спроба відкрити файл невдала, то `fopen` повертає значення `NULL`.

Функція `fopen` пов'язує ім'я файлу із вказівником на файлову змінну. Далі вся робота з файлом ведеться через цю змінну:

```
FILE *lst;//створення файлової змінної
lst=fopen("my_file.txt", "r");//відкриття файлу my_file.txt
 ↪  для читання.
```

Способи доступу до файлу описано у табл. 10. Щоб відкрити файл у двійковому режимі, до наведених у таблиці прапорців додається "b", у текстовому – "t". Наприклад, "rb" означає, що необхідно відкрити двійковий файл для читання, "a+b" – дописати інформацію у двійковий файл, "w+t" – створити текстовий файл для запису і зчитування.

Табл. 10: Режими роботи з файлами

| Режим доступу | Пояснення | Якщо файл існує | Якщо файл відсутній |
|---|---|---|---|
| r | відкрити файл для читання | читати від початку | помилка відкривання |
| w | створити файл для запису | видалити вміст | створити новий |
| a | відкрити файл для додавання даних | дописати в кінець | створити новий |
| r+ | відкрити файл для читання і запису | читати від початку | помилка |
| w+ | створити файл для читання і запису | видалити вміст | створити новий |
| a+ | відкрити файл для читання і запису | дописати в кінець | створити новий |

Після відкриття файлу для зчитування у вказівник записується адреса початку цього файлу. При зчитуванні блоку інформації `lst` змінює значення на адресу першого непрочитаного байта. Зчитування припиняється, коли досягнуто кінця файлу.

Після закінчення роботи з файлом його потрібно закрити. Це робиться за допомогою бібліотечної функції `fclose` з прототипом

```
int fclose(FILE *lst);
```

Єдиним параметром функції є файлова змінна. Якщо завершення успішне, то функція `fclose` повертає нуль. Будь-яке інше значення свідчить про помилку.

Функції введення та виведення даних із файлів обов'язковим параметром

приймають вказівник на файл типу `FILE *`. Основні функції для роботи у текстовому режимі такі:

- `int fgetc(FILE *fp);` – читає один символ із вхідного файлу, заданого вказівником `fp`. Функція повертає значення прочитаного символу або у випадку помилки – `EOF`.
- `char *fgets(char *buf, int n, FILE *fp);` – читає $n-1$ символ із вхідного потоку, заданого `fp`. Функція копіює рядок у буфер `buf` з додаванням символу `\0` у кінець.
- `int fscanf(FILE *fp, const char *format, ...);` – читає рядок із файлу до першого пробілу. Правила форматування рядка `format` функцією `fscanf` такі самі, як і для `scanf`.
- `int fputc( int c, FILE *fp );` – записує символ `c` у файл за вказівником `fp`. У разі успішної операції функція повертає записаний символ, інакше – `EOF`.
- `int fputs( const char *s, FILE *fp );` – записує рядок у файл, пов'язаний з `fp`. Повертає невід'ємне значення у випадку успіху, інакше – `EOF` з-за будь-якої помилки.
- `int fprintf(FILE *fp,const char *format, ...);` – форматований запис у файл. За винятком необхідності задавати файлову змінну, працює аналогічно до функції `printf`.

Приклад. У файлі "input.txt" записано цілі числа, кожне з нового рядка. Знайти та записати суму цих чисел у файл "output.txt". Код програми:

```
#include <stdio.h>
int main()
{
    FILE *f = NULL;
    int n, k, s=0;
    f = fopen("input.txt", "r");//відкрили файл "input.txt"
    if (f == NULL)
        printf("Error!");
    else
        while(! feof(f))
        {
            k = fscanf(f, "%d", &n);//зчитування файлу
            s+=n;//сумування
        }
    fclose(f);//закрили файл "input.txt"
```

```
16      f = fopen("output.txt", "w");//відкрили файл "output.txt"
17      fprintf(f, "%d", s);//запис у файл "output.txt"
18      fclose(f);//закрили файл "output.txt"
19      return 0;
20  }
```

У цьому прикладі файлова змінна почергово пов'язується з файлами "input.txt" та "output.txt". У рядку 7 перевіряється успішність відкривання файлу. Якщо операція вдала, тоді у циклі `while` функція `fscanf` порядково зчитує числа з файлу, поки не буде досягнуто кінця файлу. Функція `feof` повертає ненульове значення, якщо `f` вказує на кінець файлу, і нуль – у протилежному випадку.

Для читання та запису двійкових файлів призначені окремі функції:

- `size_t fread(void *ptr, size_t s, size_t n, FILE *fp);` – зчитує з файлу `fp` `n` блоків даних розміром `s` кожний і записує їх за адресою `ptr`.
- `size_t fwrite(const void *ptr, size_t s, size_t n, FILE *fp);` – зчитує, починаючи з адреси `ptr`, `n` блоків даних розміром `s` кожний та записує їх у файл за вказівником `fp` .

Розглянемо приклад запису структур у файл.

```
1   #include <stdio.h>
2   struct tpoint{int x,y,z;};
3   int main()
4   {
5       struct  tpoint s[2] = {10,20,30,40,50,60};
6       FILE *f_in, *f_out;
7       f_in = fopen("library.txt", "wb");
8       fwrite(&s,sizeof(struct tpoint),2,f_in);//запис структур у
    ↪   файл
9       fclose(f_in);//
10      struct  tpoint s_read;
11      f_out = fopen("library.txt", "rb");
12      fread(&s_read,sizeof(s_read), 1, f_out);//читання
    ↪   структури з файлу
13      printf("%d %d %d", s_read.x, s_read.y, s_read.z);
14      fclose(f_out);
15      return 0;
16  }
```

У файл `library.txt` записується масив структур `s` типу `tpoint` із двох еле-

ментів. Функції `fwrite` передається адреса початку масиву `&s`, розмір одного елемента масиву `sizeof(struct tpoint)` та розмір масиву – 2 об'єкти (рядок 8). Якщо функції `sizeof` отримує не назву типу, а ім'я змінної, то записати масив структур `s` у файл можна так:

```
fwrite(&s,sizeof(s),1,f_in);
```

Зчитування одної структури відбувається за командою у рядку 12. У програмі з метою спрощення коду опущена перевірка коректності відкривання файлів.

## Список рекомендованої літератури


- Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. С. 117-151.
- Шпак З.Я. Програмування мовою С. Львів : Оріяна-Нова, 2006. С. 164-182, 311-327.
- Пінчук В.П., Лозовська Л.І. Програмування мовою С/С++ з прикладами та вправами : навчальний посібник. Запоріжжя : ЗНТУ, 2008. С. 107-121.


## Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями до лабораторної роботи.
2. Написати програми для розв'язання задач згідно з варіантом.
3. Оформити звіт до роботи.

### Варіанти завдань

### Варіант 1

1. Написати програму, яка записує у файл інформацію про особисту колекцію книголюба. Структура запису: шифр книги, автор, рік видання, місцезнаходження (номер стелажу, шафи та полиці). Кількість записів довільна.

2. Написати програму, яка читає файл, створений у попередньому завданні, і за вибором користувача виводить на екран таку інформацію:

- місцезнаходження книги автора A назви B;

- список книг автора C, які знаходяться в колекції;
- кількість книг видання X року, які знаходяться в колекції.

Значення A, B, C, X ввести з клавіатури.

**Варіант 2**

1. Написати програму, яка створює файл-довідник із даними про транзистори. Структура запису: марка, провідність ($n$-$p$-$n$, $p$-$n$-$p$), максимальна розсіювана потужність, максимальна робоча частота. Кількість записів довільна.

2. Написати програму, яка читає файл створений у попередньому завданні, і за вибором користувача виводить на екран таку інформацію:

- всі дані для заданої марки транзистора;
- за введеним із клавіатури значенням максимальної розсіюваної потужності та максимальної робочої частоти виводить усі транзистори, параметри яких збігаються із заданими;
- вивести інформацію про всі комплементарні пари транзисторів (у яких провідність різна, а інші параметри однакові).

Для побудови меню вибору варіанта використати оператор `switch`.

**Варіант 3**

1. Написати програму, яка заносить у файл інформацію про співробітників університету. Структура запису: прізвище працівника, назва кафедри, рік народження, стаж роботи, посада. Кількість записів довільна.

2. Написати програму, яка читає файл, створений у попередньому завданні, і за вибором користувача виводить на екран таку інформацію:

- список працівників пенсійного віку на сьогоднішній день із зазначенням стажу роботи;
- середній стаж працівників відділу “X”;

**Варіант 4**

1. Написати програму, яка записує у файл інформацію про пацієнтів лікарні: прізвище пацієнта, стать, вік, місце проживання (місто), діагноз. Кількість записів довільна.

2. Написати програму, яка читає файл, створений у попередньому завданні, і за вибором користувача виводить на екран таку інформацію:

- кількість пацієнтів, які прибули до клініки з іншого міста;

- список пацієнтів, старших X років з діагнозом Y. Значення X, Y ввести з клавіатури.

**Варіант 5**

1. Написати програму, яка створює файл, що містить інформацію про розклад телепрограм на день для кількох каналів (3-4). Структура запису: канал, назва програми, час початку програми, час закінчення програми.

2. Написати програму, яка читає файл, створений у попередньому завданні, і за вибором користувача виводить на екран наступну інформацію:

- усю програму телепередач на день для заданого каналу;
- назву найдовшої та найкоротшої (за тривалістю) телепрограми;
- за заданим із клавіатури часом видати назви програм, які транслюватимуться у цей час.

Для побудови меню вибору варіанта використати оператор `switch`.

## Запитання та завдання для самоперевірки

1. Наведіть визначення структури?
2. Що називають об'єднанням (union)?
3. У чому відмінність між об'єднанням і структурою?
4. Що таке файловий тип даних?
5. Які існують типи файлів?
6. Які операції з файлами ви знаєте?
7. У яких режимах можна відкрити файл?

# Лабораторна робота 7
# Динамічна пам'ять

**Мета роботи** – отримати навички з алгоритмізації та програмування задач із використанням динамічної пам'яті.

## Теоретичні відомості

Будь-яка змінна будь-якого типу розміщується в пам'яті. Вказівник на змінну зберігає місце її розташування (тобто адресу) в пам'яті комп'ютера. Важлива особливість мови C полягає в тому, що вказівники в ній типізовані. Це означає, що, коли змінна, наприклад `p`, має тип "вказівник на `int`", то значеннями змінної `p` можуть бути адреси лише змінних типу `int` та не можуть бути адреси змінних типу `double` чи інших. Взагалі, який би тип у мові C не розглянути (позначимо його T), йому відповідає свій тип вказівника – вказівник T. Щоб оголосити змінну типу вказівника на деякий тип, потрібно перед іменем змінної поставити зірочку:

`ім'я типу *ім'я_змінної_вказівника;`

В одному оголошені можна поєднувати звичайні змінні та змінні одного типу вказівника. Наприклад:

```
int *p;
double *q1, *q2;
int x, *r, *t;//оголошено змінну x та два вказівника r і t
```

Операція отримання адреси "`&`" знаходить адресу змінної. Нехай `a` – ім'я деякої змінної, тоді вираз `&a` дає адресу змінної `a`. Наприклад, після оператора присвоєння

```
int *p, x = 1;
p = &x;
```

значенням вказівника `p` стане адреса змінної `x`.

Операція розіменування вказівника дає змогу знайти значення за адресою, на яку він показує. Позначається розіменування зірочкою перед іменем змінної-вказівника. Якщо `p` – ім'я деякої змінної-вказівника, то вираз `*p` дає

змінну, адресу якої містить `p`.

Варто пам'ятати, що вказівники не лише відкривають нові можливості програмування, але при неправильному використанні є джерелом помилок. Типовою помилкою є використання вказівника, якому не присвоєно значення адреси деякого об'єкта, але з яким програма намагається працювати, наче із вказівником на реальні дані. Наприклад, після виконання коду

```
1 int y, *p;
2 y = *p;
3 *p = 1;
```

значення `y` не визначене, незважаючи на те, що дані командою `*p = 1` за адресою у вказівнику буде записано число "1".

### Динамічна пам'ять

Програми, які розглядалися раніше, працюють зі заздалегідь відомими та зафіксованими в тексті програми розмірами масивів. Кількість елементів в оголошеннях на кшталт `int a[10];` відома ще на етапі трансляції програми і не може бути змінена динамічно, в процесі її виконання. Крім того, виділення та вивільнення пам'яті здійснювалося автоматично – програміст не має контролю за цим процесом.

Засоби динамічного управління пам'яттю дозволяють значно розширити можливості програмування та принципово збільшують потужність мови. Програма отримує можливість під час виконання самостійно визначати потрібний розмір пам'яті, створювати та видаляти масиви даних.

Динамічне управління пам'яттю в мові C здійснюється за допомогою стандартних функцій, оголошення яких містяться у файлі *stdlib.h*.

Робота з динамічними масивами здійснюється через змінну типу вказівника. Після створення масиву ця змінна міститиме адресу його першого елементу.

Для виділення шматка динамічної пам'яті використовується функція `malloc`. Вона приймає один аргумент – розмір у байтах області пам'яті `sizemem` цілого типу `size_t`, яку потрібно виділити:

```
void *malloc(size_t sizemem);
```

Функція `malloc` виділяє блок пам'яті, розміром `sizemem` байт, і повертає вказівник на початок блоку. Зміст виділеного блоку пам'яті не ініціалізується, а залишається з невизначеними значеннями.

Обчислити розмір масиву можна, помноживши кількість елементів на розмір у байтах одного елемента. Наприклад, виділити пам'ять під масив з 10 цілих чисел можна так:

```
int *p, n = 10;
p = (int *) malloc(n*sizeof(int));
```

Оскільки `malloc` повертає вказівник на `void *`, то необхідне приведення типу виділеної пам'яті до типу елементів масиву. У розглянутому випадку це виконується командою `(int *)`.

Виділити блок пам'яті, заповнений нулями, можна функцією `calloc`, що має сигнатуру

```
void *calloc(size_t number, size_t size);
```

Функція `calloc` виділяє блок пам'яті для масиву розміром `number` елементів, кожен із яких займає `size` байт, і ініціалізує всі біти нулями. Об'єм виділеної пам'яті становитиме `number*size` байт. Код для створення динамічного масиву розміром 10 цілих чисел із використанням `calloc` буде такий:

```
int *p, n = 10;
p = (int *) calloc(n, sizeof(int));
```

Використання динамічних даних покладає на програміста відповідальність як за виділення, так і за звільнення пам'яті. Якщо дані не потрібні, їх необхідно видаляти. Іншою небезпекою роботи з динамічною пам'яттю є можливість зміни значення вказівника на блок пам'яті. Наприклад, при виконанні коду

```
int *p, n = 10;
p = (int *) calloc(n,sizeof(int));
p =&n;
```

вказівнику присвоєно нове значення. При цьому доступ до виділеного масиву буде втрачено, і зберегти інші дані у цій пам'яті буде неможливо. Накопи-

чення таких ситуацій призводить до витоку пам'яті (memory leak), при якому відбувається постійне зменшення доступної програмі оперативної пам'яті. Звільнення пам'яті, втраченої в результаті витоку, виконується при завершенні роботи програми.

Для звільнення пам'яті призначена функція `free`:

```
void free(void *ptrmem);
```

Блок пам'яті, раніше виділений за допомогою виклику `malloc`, `calloc` або `realloc`, звільняється і може використовуватися програмами або ОС. Функція `free` залишає значення вказівника `ptrmem` незмінним. Отже він, як і раніше, вказує на той самий блок пам'яті, а не на нульовий вказівник.

Якщо необхідно змінити розмір раніше виділеного блоку, то доцільно використати функцію `realloc`:

```
void *realloc( void *ptrmem, size_t size );
```

Функція `realloc` виконує перерозподіл блоків пам'яті. Розмір блоку, на який посилається параметр `ptrmem`, змінюється до `size` байт. Ця функція може переміщати блок пам'яті на нове місце, у цьому випадку функція повертає вказівник на нове місце в пам'яті. Дані в пам'яті зберігаються, навіть якщо новий блок має менший розмір, ніж старий. Відкидаються тільки ті дані, які не вмістилися в новий блок. Якщо нове значення `size` більше за старе, то вміст знову виділеної пам'яті буде невизначеним. Якщо `ptrmem` дорівнює `NULL`, функція `realloc` поводиться так само, як `malloc`, тобто виділяє пам'ять і повертає вказівник на цю ділянку пам'яті. У разі, коли `size` дорівнює 0, раніше виділена пам'ять звільняється, як при виклику функції `free`, і повертається нульовий вказівник.

### Динамічні дані

Динамічні дані пов'язані з програмою за допомогою адреси в пам'яті і не є локальними для якоїсь функції. Створення, оброблення та видалення динамічного об'єкта може відбуватися в різних функціях, яким відома його адреса. На відміну від локальних даних, динамічні не видаляються після завершення роботи функції, а тому можуть бути передані (чи повернуті) в інше

місце програми через повернення вказівника. Час життя динамічно створеного об'єкта – це час від його створення будь-якою з функцій `malloc`, `alloc` або `realloc` до видалення функцією `free`.

Проілюструємо сказане прикладом програми для розрахунку чисел Фібоначі.

```
#include <stdio.h>
#include <stdlib.h>
long int *alloc_array(int n);
void kill_array(long int *p);
long int *make_fibonacci_table(long int *p, int n);
int main ()
{
    long int *fibon ;
    int m, i;
    printf ( "Кількість наближень?\n");
    scanf ("%d", &m);
    fibon = alloc_array (m);//виділити пам'ять
    make_fibonacci_table (fibon, m);//порахувати
    ↪ послідовність
    for (i = 0; i < m; ++i)//вивести послідовність
        printf ("%16ld\n", fibon[i]);
    kill_array(fibon);//звільнити пам'ять
return 0;
}
long int *alloc_array (int n)
{
    return (long int *) malloc(n * sizeof(long int));
}
    void kill_array(long int *p)
{
    free(p);
}
long int *make_fibonacci_table(long int *p, int n)
{
    int i;
    p[0] = 1;
    p[1] = 1;
    for (i = 2; i < n; ++i)
    p[i] = p[i - 1] + p[i - 2];
    return p;
}
```

Функція `alloc_array` виділяє пам'ять для масиву з `n` цілих чисел та повертає вказівник на нього. Усередині викликається бібліотечна функція `malloc`.

Функція `make_fibonacci_table`, яка заповнює масив числами Фібоначі, приймає два аргументи: вказівник на масив цілих чисел та кількість елементів у ньому.

Функція `kill_array` звільняє динамічно виділену пам'ять.

Обмін динамічним масивом `fibon` між `main`, `alloc_array`, `make_fibonacci_table` та `kill_array` ведеться через передавання вказівника на дані. Отже, масив, створений динамічно в одній функції, обробляється в інших функціях. Для цього перша функція має просто повернути вказівник на створений нею масив.

### Вказівники вищих рівнів

Для будь-якого типу `T` в мові C має сенс відповідний тип вказівника `T*`. Оскільки вказівник також має свій тип, то можна задати вказівник на вказівник `T**`, вказівник на вказівник на вказівник `T***` і т.д. Отже, можна отримати вказівники вищих порядків. У коді

```
int x0 = 1;
int *p1 = &x0;
int **p2 = &p1;
int ***p3 = &p2;
```

змінна `x0` містить дані – ціле число 1, а змінна `p1` зберігає адресу в пам'яті тієї комірки, де записано `x0`. Але ж `p1` – це також змінна, яка займає певне місце в пам'яті. Змінна `p2` зберігає адресу, за якою розташована змінна `p1`. Аналогічно, `p3` містить адресу, за якою міститься в пам'яті змінна `p2`. Змінні `p1`, `p2`, `p3` – це вказівники відповідно першого, другого та третього рівнів.

Одновимірний масив дійсних чисел у мові C обробляється через вказівник першого рівня `T *`, який містить адресу першого елементу. Тоді двовимірний масив (матриця), що є одновимірним масивом вказівників на одновимірні масиви, повинен оброблятися через вказівник другого рівня `T **`.

Приклад. Написати програму, яка виводить на екран таблицю множення для заданого числа.

```
#include <stdio.h>
#include <stdlib.h>
int main()
```

```
4  {
5      int **p, i, a, n=10;
6      a = 15;
7      p = (int **) malloc(n * sizeof(int *));
8      for (i = 0; i < n; i++)
9          p[i] = (int *) malloc(3 * sizeof(int));
10     for (i = 0; i < n; i++)//запис даних у матрицю
11     {
12         p[i][1]=a;
13         p[i][2]=i+1;
14         p[i][3]=p[i][1]*p[i][2];
15     }
16     for (i = 0; i < n; i++)//виведення матриці
17         printf("%d*%d = %d\n", p[i][1], p[i][2], p[i][3]);
18     for (i = 0; i < n; i++)//звільнення пам'яті
19         free(p[i]);
20     free(p);
21     return 0;
22 }
```

У програмі використано динамічне виділення пам'яті для матриці розміром 10*3. У рядку 5 оголошено вказівник на масив рядків p**. Виділення пам'яті виконується командою у рядку 7. У циклі for (рядки 7-8) для кожного рядка матриці окремо виділяється пам'ять для запису трьох елементів у форматі множник-множник-добуток. Звільняється пам'ять у оберненому порядку: спочатку для кожного рядка матриці у циклі (рядки 18-19), потім для масиву вказівників на рядки (рядок 20).

## Список рекомендованої літератури


- Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. С. 96-101, 127-151.
- Шпак З.Я. Програмування мовою С. Львів : Оріяна-Нова, 2006. С. 246-253.
- Пінчук В.П., Лозовська Л.І. Програмування мовою С/С++ з прикладами та вправами : навчальний посібник. Запоріжжя : ЗНТУ, 2008. С. 67-72.

## Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом до лабораторної роботи.
2. Вивчити стандартні функції, які забезпечують основні операції з динамічною пам'яттю.
3. Виконати завдання згідно з варіантом, використовуючи динамічні масиви. Вхідний масив ввести з клавіатури, заповнити алгоритмічно або згенерувати у випадковий спосіб (у залежності від постановки задачі). У програмі передбачити можливість введення з клавіатури розмірності оброблюваного масиву в діапазоні оголошеної максимальної розмірності. Основні етапи роботи з масивом (введення з клавіатури або ініціалізація, виведення на екран, роботу з масивом) оформити у вигляді функцій. Індексацію елементів масиву здійснювати за допомогою вказівників. Вхідний масив і отриманий результат вивести на екран, а також записати в текстовий файл.
4. Оформити звіт до роботи.

### Варіанти завдань

#### Варіант 1

1. В одновимірному масиві з $n$ елементів знайти порядкові номери першого від'ємного й останнього додатного елементів (якщо такі є). Значення елементів і їх порядкові номери вивести на екран або видати відповідне повідомлення.

2. У матриці $A_{n\times n}$ знайти перший стовпець, який не містить від'ємних елементів, і помножити його як вектор на матрицю A.

#### Варіант 2

1. Ввести одновимірний масив з $n$ елементів. Обчислити окремо суми всіх від'ємних і додатних чисел, їх кількість.

2. Заповнити матрицю $A_{n\times n}$ по спіралі масивом натуральних чисел $1, 2, ..., n \times n$, починаючи з елемента, що знаходиться у верхньому лівому кутку.

#### Варіант 3

1. Перевірити характер послідовності з $n$ цілих чисел: спадна, неспадна, зростаюча, незростаюча. Вивести послідовність та відповідне повідомлення.

2. Розташувати рядки матриці $A_{n\times n}$ в порядку убування модулів сум елементів рядків.

**Варіант 4**

1. Ввести одновимірний масив з $n$ елементів. Сформувати на його місці новий масив, у якому першим елементом буде останній елемент старого, другим – передостанній і т.д.

2. Із матриці $A_{n\times n}$ отримати на тому ж місці матрицю, викресливши з початкової матриці рядок і стовпець, на перетині яких знаходиться максимальний елемент. Якщо елементів декілька, то обрати перший максимальний.

**Варіант 5**

1. Елемент називається локальним мінімумом (максимумом), якщо у нього немає сусіда меншого (більшого), ніж він сам. Знайти всі локальні мінімуми і максимуми в заданому масиві з $n$ елементів.

2. Заповнити матрицю $A_{n\times n}$ масивом випадкових цілих чисел. У кожному стовпці знайти максимальний елемент та обміняти його з елементом головної діагоналі, який належить цьому стовпцю.

## Запитання та завдання для самоперевірки

1. Як оголосити вказівник на $n$-вимірний масив?
2. Як отримати доступ до масиву в динамічній пам'яті, що виділений іншою функцією?
3. У чому полягає різниця між динамічним та статичним масивом?
4. Чи можна розширити динамічну пам'ять для масиву?
5. Як звільнити пам'ять, виділену під багатовимірний масив?
6. Чи може функція `realloc` замінити `malloc` і `free`?

# Лабораторна робота 8
# Сортування і пошук

**Мета роботи** – отримати практичні навички оброблення масивів із застосуванням алгоритмів сортування та пошуку.

## Теоретичні відомості

Алгоритми сортування застосовують для впорядкування даних за певною ознакою (зростання або спадання, ключем і т.п.). Залежно від часу виконання та необхідної для роботи пам'яті розрізняють часову та ємнісну (просторову) складність алгоритму.

*Часова складність* оцінюється як час роботи алгоритму з великим (у теорії безкінечним) набором вхідних даних. Розрізняють найкращу, середню та найгіршу тривалість виконання, які позначають функціями, що залежать від об'єму вхідних даних. Наприклад, час роботи алгоритму $O(N^2)$ означає, що для вхідних даних розміром $N$ тривалість роботи алгоритму пропорційна до $N^2$ і знаходиться в межах $C_1N^2 < O(N^2) < C_2N^2$, де $C_1$, $C_2$ – константи.

*Ємнісна складність* визначається вимогами до необхідного обсягу пам'яті при роботі алгоритму.

*Стабільним* (або *стійким*) називається такий алгоритм сортування, що не змінює порядок елементів із однаковим ключем (значенням).

Основні характеристики алгоритмів, які вивчаються у лабораторній роботі, наведено у табл. 11

Табл. 11: Швидкодія методів сортування

| Метод сортування | Найгірша швидкодія | Середня швидкодія | Найкраща швидкодія | Стабільний |
|---|---|---|---|---|
| Бульбашки | $O(N^2)$ | $O(N^2)$ | $O(N)$ | + |
| Вибором | $O(N^2)$ | $O(N^2)$ | $O(N^2)$ | + |
| Включенням | $O(N^2)$ | $O(N^2)$ | $O(N)$ | + |
| Шелла | $O(N^2)$ | | $O(N \log N)$ | - |
| Швидке | $O(N^2)$ | $O(N \log N)$ | $O(N \log N)$ | - |

### Сортування “бульбашкою”

Сортування “бульбашкою” (bubble sort) ґрунтується на порівнянні та перестановці сусідніх елементів. Переміщення елементів масиву здійснюється так: масив переглядається справа наліво (з кінця до початку), при цьому порівнюються пари сусідніх елементів; якщо елементи в парі розміщені в порядку убування, вони лишаються без змін, а якщо ні – міняються місцями. У результаті першого проходу найменше число буде поставлене на початок масиву. Власне тому метод отримав таку назву: “легші” елементи поступово переміщуються на “верх” (початок), “важчі” – на “дно” (у сторону кінця масиву). У другому проході операції виконуються над елементами з $N-1$-ого до другого. За кожний прохід один елемент “знаходить” своє місце у відсортованій частині масиву. Після $n$-го проходу буде впорядковано $n$ елементів. Якщо при проході масиву не виконається жодної перестановки елементів масиву, це означатиме, що масив уже відсортований і процес сортування можна зупинити. Факт перестановки фіксується за допомогою деякої змінної (flag), яка змінює значення з “0” на “1”, коли виконується перестановка в будь-якій парі. Код, що реалізує сортування “бульбашкою” масиву цілих чисел `a` розміром `N`:

```
int i = 0, j = 0, flag = 0, c, N;
int a[N]; //масив
/* ... */
do
{
    flag = 0; //скинути прапорець
    for (j = N - 2; j >= i ; j--)
        if (a[j] > a[j + 1])
        {
            c = a[j];
            a[j] = a[j + 1];
            a[j + 1] = c;
            flag = 1;  //встановити прапорець
        }
    i++;
}
while (flag); //вихід, якщо flag = 0
```

Приклад перетворення масиву у процесі сортування наведено у табл. 12.

Табл. 12: Сортування “бульбашкою”

| Початковий масив | 43 | 10 | 48 | 45 | 73 | 81 | 90 | 85 | 24 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|
| Перший перегляд масиву | **10** | 43 | 18 | 48 | 45 | 73 | 81 | 90 | 85 | 24 |
| Другий перегляд масиву | **10** | **18** | 43 | 24 | 48 | 45 | 73 | 81 | 90 | 85 |
| Третій перегляд масиву | **10** | **18** | **24** | 43 | 45 | 48 | 73 | 81 | 85 | 90 |
| Четвертий перегляд масиву | **10** | **18** | **24** | **43** | **45** | **48** | **73** | **81** | **85** | **90** |

### Сортування методом вибору

При сортуванні методом вибору (selection sort) знаходиться мінімальний елемент масиву, і він міняється місцями з першим елементом. Далі у решті масиву знову знаходиться мінімальний елемент і міняється місцем з другим. Даний процес продовжується доти, поки не буде поставлений на місце $N-1$-й елемент. Процес сортування нескладно зрозуміти з прикладу в табл. 13. Елементи, які змінили свої позиції, позначені жирним шрифтом.

Табл. 13: Сортування методом вибору

| Початковий масив | 37 | 84 | 5 | 68 | 48 |
|---|---|---|---|---|---|
| Перший перегляд масиву | **5** | 84 | **37** | 68 | 48 |
| Другий перегляд масиву | 5 | **37** | **84** | 68 | 48 |
| Третій перегляд масиву | 5 | 37 | **48** | 68 | **84** |
| Четвертий перегляд масиву | 5 | 37 | 48 | 68 | 84 |

У порівнянні із сортуванням “бульбашкою”, при сортуванні вибором у більшості випадків необхідна менша кількість обмінів елементів: за один прохід потрібно не більше однієї перестановки, проте вимагає $N-1$-го проходу масиву.

Реалізувати сортування вибором можна за допомогою такого коду:

```
int i = 0, j = 0, c, n_min, N = 5;//N - розмір масиву
int a[N]; //масив
/* ... */
for(i = 0; i < N-1; i++)
{
    n_min = i ;
    for (j =  i+1; j < N; j++)//пошук мінімального у решті масиву
        if(a[j] < a[n_min])//якщо є менший за поточний, тоді запам'ятаємо його індекс
            n_min = j;
```

```
10
11      if(n_min != i)//обмін, якщо знайдено менший
12      {
13          c = a[i];
14          a[i] = a[n_min];
15          a[n_min] = c;
16      }
17  }
```

### Сортування включенням

Сортування включенням (insertion sort) полягає у "включенні" на $i$-му проході масиву елемента $a[i]$ в потрібну позицію серед уже впорядкованих елементів $a[1]$, ..., $a[i-1]$. Після такого включення перші $i$ елементів будуть відсортовані. Реалізація сортування включенням може бути такою:

```
1   int i, j, temp, N = 5;
2   int a[N];
3   /* ... */
4   for (i = 1; i < N; i++)//
5   {
6       j = i;
7       temp = a[i];//елемент для включення
8       while (j > 0 && temp < a[j - 1])//пошук місця включення
9       {
10          a[j] = a[j - 1];
11          j--;
12      }
13      a[j] = temp;
14  }
```

Значення кожного елемента масиву, починаючи з другого (`i = 1`, цикл `for`), записується у змінну `temp`, яка далі порівнюється з елементами з меншим індексом (`temp` послідовно порівнюється з елементами `a[i - 1]`, `a[i - 2]`, і доти, поки для чергового елемента `a[j]` виконується співвідношення `a[j] > temp`, елементи `a[j]` і `a[i]` міняються місцями (цикл `while`). Якщо вдасться зустріти такі елементи, що `a[j] < temp`, або, якщо досягнута нижня межа масиву (`j = 0`), тоді здійснюється перехід до опрацювання елемента `a[i + 1]`, доки не буде досягнута верхня межа масиву. Приклад роботи алгоритму наведено у табл. 14.

Табл. 14: Сортування включенням

| | | | | | |
|---|---|---|---|---|---|
| Початковий масив | 29 | 17 | 25 | 21 | 1 |
| Перший перегляд масиву | 17 | 29 | 25 | 21 | 1 |
| Другий перегляд масиву | 17 | 25 | 29 | 21 | 1 |
| Третій перегляд масиву | 17 | 21 | 25 | 29 | 1 |
| Четвертий перегляд масиву | 1 | 17 | 21 | 25 | 29 |

### Швидке сортування

Швидке сортування (quicksort) – широко відомий алгоритм сортування, розроблений англійським ученим Чарльзом Хоаром. Це один із найшвидших відомих універсальних алгоритмів сортування масивів, хоча він і має ряд недоліків. Алгоритм передбачає три кроки:

1. Вибирається опорний елемент масиву.
2. Масив розбивається на два – лівий і правий щодо опорного елемента. Далі масив перетворюється так, щоб усі елементи, менші за опорний, виявилися зліва від нього, а всі елементи, більші за опорний, – праворуч від нього.
3. Далі рекурсивно повторюється крок 2 для кожного з двох новостворених масивів. Кожен раз при повторенні перетворення чергова частина масиву розбивається на дві менші і т. д., поки не вийде масив із двох елементів.

Реалізувати описаний алгоритм можна рекурсивним викликом функції `quick_sort`:

```
void quick_sort(int left, int right, int *x)
{
    int i;
    if(left >= right) return;
        i = partition(left, right, x);
    quick_sort (left, i - 1, x);
    quick_sort (i+1, right, x);
}
int partition(int left, int right, int *x)
{
    int base,i, j,temp;
    base = x[right]; //опорний елемент підмасиву
    i = left - 1;//індекс останнього елемента не меншого за
 ↪  опорний
    for(j = left; j <= right - 1; j++)
```

```
15      {
16          if(x[j] <= base)//шукаєти елементи не більші за
    ↪   опорний
17          {
18              i++;
19              if (i != j) //обмінятиі місцями, якщо це різні
    ↪   елементи
20              {
21                  temp = x[i];
22                  x[i] = x[j];
23                  x[j] = temp;
24              }
25          }
26      }
27      //ставим на місце опорний елемент
28      temp = x[i + 1];
29      x[i + 1] = x[right];
30      x[right] = temp;
31      return i + 1;
32  }
```

Завдання функції `partition` – пошук правильного місця для опорного елемента, знайдена позиція повертається як результат роботи. У поданій реалізації за опорний (`base`) приймається останній елемент у невідсортованій частині масиву `x`, обмеженій індексами `left` і `right`. Приклад перетворення масиву при першому проході наведено у табл. 15.

Табл. 15: Швидке сортування

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | $i$, $j$ |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 3 | 4 | 5 | 1 | 6 | 2 | 7 | 5 | масив для сортування |
| 9 | 3 | 4 | 5 | 1 | 6 | 2 | 7 | 5 | $j = 0$, $9 \leq 5$? – false |
| 3 | 9 | 4 | 5 | 1 | 6 | 2 | 7 | 5 | $j = 1$, $3 \leq 5$? – true, i=0; swap(9, 3) |
| 3 | 4 | 9 | 5 | 1 | 6 | 2 | 7 | 5 | $j = 2$, $4 \leq 5$? – true, i = 1; swap(4,9) |
| 3 | 4 | 5 | 9 | 1 | 6 | 2 | 7 | 5 | $j = 3$, $5 \leq 5$? – true, i = 2; swap(5,9) |
| 3 | 4 | 5 | 1 | 9 | 6 | 2 | 7 | 5 | $j = 4$, $1 \leq 5$? – true, i = 3; swap(1,9) |
| 3 | 4 | 5 | 1 | 9 | 6 | 2 | 7 | 5 | $j = 5$, $6 \leq 5$? – false |
| 3 | 4 | 5 | 1 | 2 | 6 | 9 | 7 | 5 | $j = 6$, $2 \leq 5$? – true, i = 4; swap(2,9) |
| 3 | 4 | 5 | 1 | 2 | 6 | 9 | 7 | 5 | $j = 7$, $7 \leq 5$? – false |
| 3 | 4 | 5 | 1 | 2 | 5 | 9 | 7 | 6 | swap(6,5) |
| 3 | 4 | 5 | 1 | 2 |  | 9 | 7 | 6 | два підмасиви сортуються окремо |

Табл. 16: Сортування Шелла

| Початковий масив | 49 | 42 | 3 | 28 | 15 | 17 | 34 | 44 | 17 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Масив після першого проходу, step = 5, 4 обміни | 17 | 34 | 3 | 17 | 9 | 49 | 42 | 44 | 28 | 15 |
| Масив після другого проходу, step = 3, 5 обмінів | 15 | 9 | 3 | 17 | 34 | 28 | 17 | 44 | 49 | 42 |
| Масив після третього проходу, step = 2, 3 обміни | 3 | 9 | 15 | 17 | 17 | 28 | 34 | 42 | 49 | 44 |
| Масив після четвертого проходу, step = 1, 1 обмін | 3 | 9 | 15 | 17 | 17 | 28 | 34 | 42 | 44 | 49 |

### Сортування методом Шелла

Головний недолік сортування включенням – обмін ведеться між сусідніми елементами. Тому бажано робити якомога ширші обміни. Алгоритм Шелла узагальнює сортування включенням із відстанями, що зменшуються. У цьому методі виконується кілька впорядкувань включенням, але кожен раз порівнюються і переставляються елементи, що розташовані на різній відстані один від одного. Після цього весь масив сортується разом звичайним методом включення. За рахунок попередніх етапів він виявляється вже близьким до відсортованого, тому обмінів потрібно вже не так багато. Візьмемо, для прикладу, масив:

```
49  42  3  28  15  17  34  44  17  9
```

Якщо задати кроки вибору підмасивів як `step[]={5,3,2,1}`, то на першому проході сортуватимуться масиви, отримані з початкового з кроком 5: `{49,17}`,`{42,34}`,`{3,44}`,`{28,17}`,`{15,9}`. У результаті буде здійснено чотири обміни. Подальший хід процесу сортування можна простежити з табл. 16. У цілому для сортування заданого масиву методом Шелла необхідно виконати 13 перестановок. Для порівняння, сортування цього ж масиву методом включення вимагає 27 обмінів.

Швидкодія сортування Шелла залежить від кроків вибору підмасивів на попередніх етапах. Якщо крок спочатку задати як половину розміру масиву і зменшувати його у два рази на кожному проході, тоді процедуру сортування можна описати так:

```
void ShellSort(int a[], int n)
{
    int step = n / 2;
    int temp;
    while (step > 0)
    {
        for(int i = 0; i < n-step; i++)
        {
            int j = i;
            while (j >= 0 && a[j] > a[j + step])
            {
                temp = a[j];
                a[j] = a[j + step];
                a[j + step] = temp;
                j -= step;
            }
        }
        step /= 2;
    }
}
```

Для заданої у табл. 16 послідовності при сортуванні з таким динамічним кроком треба виконати 16 обмінів.

### Пошук елементів у масивах

*Лінійний пошук* – алгоритм послідовного пошуку заданого значення ключа серед елементів невідсортованого масиву. Шукане значення може бути як першим, так і останнім елементом у масиві, але в середньому програма повинна порівняти з ключем пошуку половину елементів масиву. Метод лінійного пошуку добре працює на невеликих або несортованих даних. Функцію для знаходження елемента `key` у масиві `a` розміром `n` можна написати так:

```
int LinearSearch (int a[], int n, int key)
{
    for (int i = 0; i < n; i++)
        if (a[i] == key)
            return i;
    return -1;
}
```

Функція повертає індекс шуканого елемента, якщо його виявлено, і значення `-1`, якщо елемент відсутній.

*Двійковий (бінарний) пошук* – ефективний алгоритм пошуку елемента у відсортованому масиві, який полягає в порівнянні шуканого значення із серединним елементом і повторенням алгоритму для однієї з половин залежно від результату порівняння.

Ітеративний алгоритм пошуку буде таким:

```
int BinarySearch(int a[], int N, int key)
{
    int left = 0, right = N - 1, midle;
    while(right >= left)
    {
        midle = (left + right) / 2;
        if(a[midle] < key)
            left = midle + 1;
        else
        if(a[midle] > key)
            right = midle - 1;
        else
            return midle;//знайдено елемент
    }
    return -1;//не знайдено такого элемента
}
```

Пошук заданого елемента ведеться у частині масиву, обмеженій індексами `left` і `right`.

### Час виконання коду

У бібліотеці *time.h* зібрано засоби для роботи з датами та часом. Визначити час виконання певного набору команд можна за допомогою функції `clock`, яка повертає кількість тактів процесора від початку роботи програми:

```
#include <time.h>
// ...

clock_t start = clock();
//набір команд
clock_t end = clock();
double cpu_time_used = ((double) (end - start)) /
 ↪  CLOCKS_PER_SEC;
```

Змінні `start` і `end` містять інформацію про кількість тактів процесора до і після виконання деяких команд. Тривалість роботи коду в секундах

`cpu_time_used` можна знайти як різницю `end - start`, нормовану до кількості тактів процесора за секунду (`CLOCKS_PER_SEC`).

## Список рекомендованої літератури

- Алгоритми і структури даних : навчальний посібник / Т.О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. С. 88-121.
- Структури даних та алгоритми : навчальний посібник / Михальов О.І., Крамаренко В.В., Ялова К.М., Новікова К.Ю. Дніпродзержинськ : ДДТУ, 2010. С. 197-222.

## Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями. За потреби скористатися додатковою літературою.
2. Розробити програму згідно з варіантом. Для сортування і пошуку повинні бути розроблені окремі функції.
3. Згенерувати масив розміром 20-50 елементів і вивести його на екран (використати при цьому власну функцію для виведення масиву на екран).
4. Відсортувати масив, обравши вказані методи та параметри сортування. Вивести впорядкований масив.
5. Здійснити пошук вказаного (з клавіатури) елемента у масиві, використовуючи вказаний метод пошуку.
6. Написати програму, яка дає змогу дослідити та порівняти ефективність сортування двома, заданими у задачі 1, алгоритмами. Визначити час сортування кожного алгоритму для довжин масивів N = 10, 100, 1000, 10000, 100000, 200000(1000000). Побудувати залежність $t = f(N)$ на одному графіку для різних методів сортування (у табличному і графічному виглядах). Структура програми повинна бути такою:

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

void sort_method_1(//your code);//sort_method_1 замінити на
 ↪  свій метод
```

```
void sort_method_2(//your code);//sort_method_2 замінити на свій метод

int *alloc_array(int N);//функція виділяє пам'ять для масиву
void kill_array(int *p);//фукція звільняє пам'ять масиву
int *make_array(int *a, int N, int M);//функція заповнює масив псевдовипадковими числами
int *copy_array(int *a, int N);//функція створює та повертає динамічну копію масиву
int main()
{
    int n[] = {10, 100, 1000, 10000, 100000, 200000};
    int m = 100;//m-1 - максимальне значення випадкового числа в масиві
    for (int i = 0; i < 6; i++)
    {
        int *a = alloc_array(n[i]);
        make_array(a, n[i], m);
        int *b = copy_array(a, n[i]);

        clock_t start = clock();
        //виклик sort_method_1
        clock_t end = clock();
        double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("Method 1, N = %d, time = %15.13lf\n", n[i], cpu_time_used);
        kill_array(a);

        start = clock();
        //виклик sort_method_2
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        printf("Method 2, N = %d, time = %15.13lf\n", n[i], cpu_time_used);
        kill_array(b);
    }
    return 0;

}
void sort_method_1(//your code)
{
    //your code
}
void sort_method_2(//your code)
{
```

```
45      //your code
46  }
47
48  int *alloc_array(int N)
49  {
50      return (int *) malloc(N *sizeof(int));
51  }
52  int *copy_array(int *a, int N)
53  {
54      //your code
55  }
56  void kill_array(int *p)
57  {
58      free(p);
59  }
60  int *make_array(int *a, int N, int m)
61  {
62      srand(time(NULL));
63      for (int i = 0; i < N; i++)//генерація  масиву для
     ↪  сортування
64      {
65          a[i] = (double)rand()/RAND_MAX *m;
66      }
67      return a;
68  }
```

Замість `//your code` необхідно написати власний код.

7. Оформити звіт до роботи.

## Варіанти завдань

### Варіант 1

1. Написати програму сортування цілих чисел за спаданням. Методи сортування: вибору та швидке.

2. Написати програму, що здійснює рекурсивний бінарний пошук елемента, введеного з клавіатури.

### Варіант 2

1. Написати програму сортування символів за зростанням. Методи сортування: вибору та вставки.

2. Написати програму, що здійснює лінійний пошук елемента, введеного з клавіатури.

### Варіант 3

1. Написати програму сортування натуральних чисел за спаданням. Методи сортування: Шелла та вибору.

2. Написати програму, що здійснює бінарний пошук елемента, введеного з клавіатури.

**Варіант 4**

1. Написати програму сортування символів з напрямком за спаданням. Методи сортування: Шелла та "бульбашки".

2. Написати програму, що здійснює лінійний пошук елемента, введеного з клавіатури.

**Варіант 5**

1. Написати програму сортування цілих чисел за зростанням. Методи сортування: швидке та "бульбашки".

2. Написати програму, що здійснює рекурсивний бінарний пошук елемента, введеного з клавіатури.

## Запитання та завдання для самоперевірки

1. Які методи сортування ви знаєте? Поясніть їх зміст.
2. Пояснити зміст лінійного та двійкового пошуку.
3. Яка перевага методу сортування Шелла?
4. За якими ознаками порівнюють методи сортування?
5. З якою метою у п. 6 завдання (рядок 20 програми) створюється копія `b` масиву `a`?

# Лабораторна робота 9
# Динамічні структури даних

**Мета роботи** – отримати практичні навички роботи з динамічними структурами даних.

## Теоретичні відомості

Структура даних задає спосіб організації та взаємодії між їх компонентами. Якщо елементи пов'язані між собою за допомогою вказівників, то такі структури називають *динамічними*. Розмір динамічних структур можна змінювати під час виконання програми.

Серед динамічних структур даних найчастіше використовуються такі: лінійні списки, стеки, черги та бінарні дерева.

*Лінійний список* – структура даних, яка є скінченною послідовністю елементів одного типу. Кількість елементів у цій послідовності називається довжиною списку. Лінійний список передбачає послідовний обхід елементів у напрямі від початку до кінця або навпаки. При роботі зі списками часто доводиться виконувати такі операції:

- додавання елемента в початок списку;
- вилучення елемента з початку списку;
- додавання елемента в будь-яке місце списку;
- вилучення елемента з будь-якого місця списку;
- об'єднання списків;
- розділення списку;
- копіювання списку;
- визначення кількості елементів списку;
- знаходження елементів з заданими властивостями;
- сортування, впорядкування елементів списку;
- перевірку, чи порожній список;
- видалення списку.

Програмна реалізація лінійний списків можлива на основі масиву або множини зв'язаних через вказівники елементів. На основі масиву реалізується

послідовне зберігання списку. При цьому необхідна окрема змінна для запису поточної кількісті елементів списку. Така організація уможливлює довільний доступ до перегляду вмісту списку та додавання нових елементів в його кінець. Але операції вставлення чи вилучення елемента в середину/із середини списку потребують зміщення всіх наступних елементів. При значних розмірах масивів вартість операцій для впорядкування списку стрімко зростає.

При зв'язаному зберіганні лінійних списків кожен елемент містить вказівник на наступний (рис. 1, *a*). Такий список називається *однозв'язним*. Робота з елементами однозв'язного списку здійснюється в напрямі від початку (голови списку) до кінця (хвоста списку), як показано на рис. 1, *b*. Якщо додати у структуру вказівник на попередній елемент, то список буде *двозв'язним* (рис. 1, *c*). Виділяють також *кільцеві* списки, у яких останній елемент зберігає адресу першого (рис. 1, *d*).

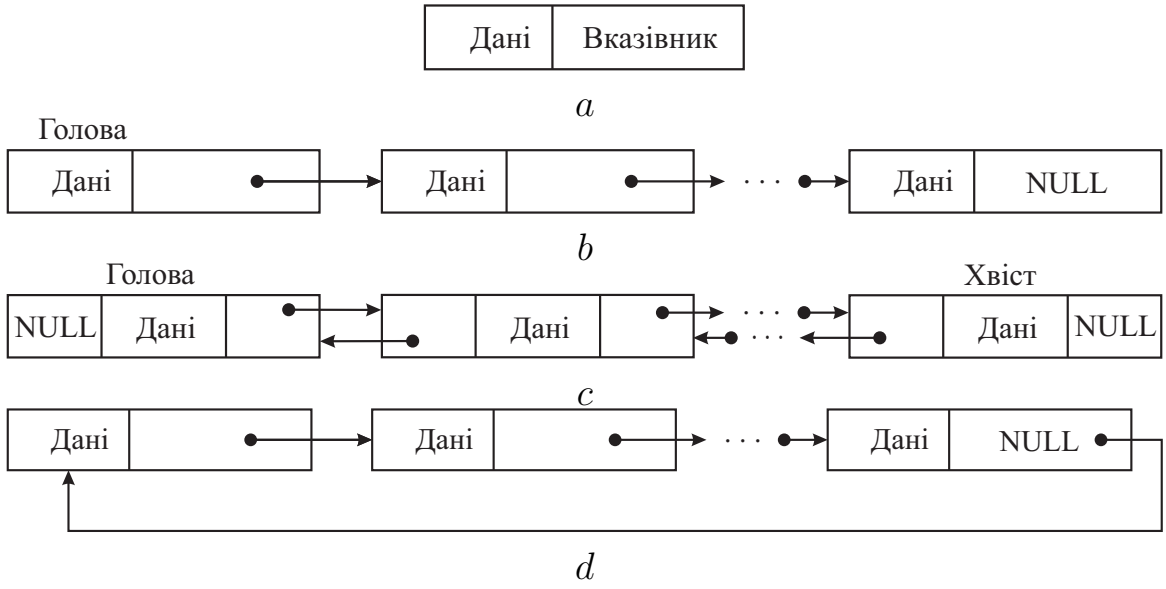


Рис. 1: Лінійні списки: *a* – структура елемента однозв'язного списку; *b* – однозв'язний список; *c* – двозв'язний список; *d* – кільцевий список

Нехай стоїть задача сформувати список слів у текстовому документі та порахувати кількість їх повторень. Тоді елемент динамічного списку таких слів повинен зберігати слово та змінну для підрахунку:

```
1  typedef struct one_word
2  {
3      char word[40];
4      int count;
5      struct one_word *next;
```

```
} pair;

typedef struct
{
    pair *head;
} word_list;
```

Структура `word_list` є списком, який містить одну змінну `head` для збереження початку. Після створення список пустий, тому `head` встановлюється в `NULL`:

```
word_list L;
L.head = NULL;
```

Щоб додати елемент до такого списку, спочатку знаходиться його хвіст:

```
int push_back(word_list *L, char word[])
{
    pair *new_word =  (pair*)malloc(sizeof(pair));//виділення пам'яті під новий елемент
    if (new_word == NULL)
        return 1;//якщо пам'ять не виділено, повертається 1

    strcpy(new_word->word, word);
    new_word->count = 1;
    new_word->next = NULL;

    if (!is_empty(L)) //перевірка, чи список не пустий
    {
        pair *current = L->head;
        while (current->next != NULL)//пошук кінця списку
            current = current->next;
        current->next = new_word;//додавання нового слова у кінець
    }
    else L->head = new_word; //якщо пустий - новий елемент стає головою
    return 0;
}
```

*Стек* – динамічна структура даних, що є впорядкованим набором елементів, у якому додавання нових елементів і видалення наявних дозволено з одного кінця – вершини стеку.

Стек реалізує принцип *LIFO* (*last in – first out*, останнім прийшов – першим пішов). Прикладом організації стеку може бути запаяна пробірка, в яку

додавати кульки і забирати їх можна тільки з однієї сторони (рис. 2). Стек можна розглядати як частковий випадок однозв'язного списку, можливості якого обмежені операціями:

- додавання елемента в стек;
- вилучення елемента зі стека;
- перевірки, чи порожній стек;
- перегляду елемента у вершині стека без видалення;
- очищення стека.

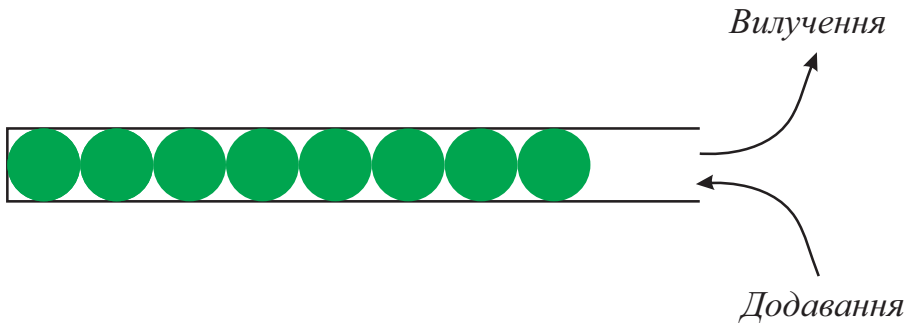


Рис. 2: Стек

Реалізуємо стек символів на базі однозв'язного списку. Структура елемента стеку може бути такою:

```
typedef struct node
{
    char data;
    struct node *next;
} node;
```

Основні функції для роботи зі стеком:

```
void push (node **head, char x)//додавання елемента
{
    node *new_node = (node *) malloc(sizeof(node));
    new_node->data = x;
    new_node->next = *head;
    *head = new_node;
}
char pop (node **head)//вилучення елемента
{
    char x;
    node *q = *head;
    x = (*head)->data;
    *head = (*head)->next;
    free(q);
```

```
15      return x;
16  }
17  int is_empty (node **head)//перевірка, чи пустий
18  {
19      if ( (*head) == NULL ) return 1;//стек пустий
20      else return 0;//стек не пустий
21  }
```

При створенні такого стеку його необхідно ініціалізувати значенням `NULL`:

```
node *stack = NULL;
```

*Черга* – це лінійний список, де елементи вилучаються з початку списку, а додаються в кінець (рис. 3) (як звичайна черга в магазині). *Двостороння черга* – це лінійний список, у якому операції додавання, вилучення і доступу до елементів можливі як спочатку, так і з кінця списку.

Черга є частковим випадком односпрямованого списку і реалізує принцип FIFO (first in – first out, першим прийшов – першим пішов). Черги створюються аналогічно до лінійних списків та стеків.

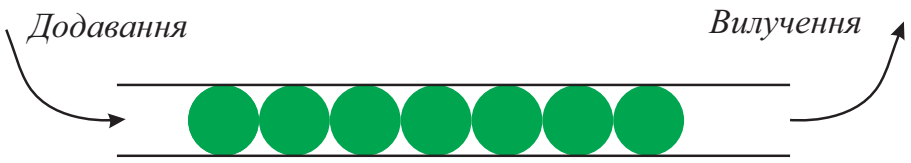


Рис. 3: Черга

Чергу зручно реалізувати як структуру, яка зберігає вказівники на перший (`head`) та останній (`tail`) елементи зв'язаного списку елементів:

```
1   typedef struct node//елемент черги
2   {
3       int  data;
4       struct node *next;
5   } node;
6
7   typedef struct//черга
8   {
9       node *head, *tail;
10  } queue;
```

Створити змінну типу черги можна так:

```
queue s;
s.head = NULL;
s.tail = NULL;
```

Приклад коду функції `push_tail` для додавання елемента до черги:

```
void push_tail(queue *q, int x)
{
    node *new_node = (node
    *)malloc(sizeof(node));//створюється новий елемент
    new_node->data = x;//у який записуються дані
    new_node->next = NULL;//
    if (q->tail)
    q->tail->next = new_node;
    q->tail = new_node;
    if (q->head == NULL)
    q->head = q->tail;
}
```

## Список рекомендованої літератури


- Алгоритми і структури даних: навчальний посібник / Т.О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. С. 34-49, 60-67.
- Структури даних та алгоритми : навчальний посібник / Михальов О.І., Крамаренко В.В., Ялова К.М., Новікова К.Ю. Дніпродзержинськ : ДДТУ, 2010. С. 31-92.
- Шпак З.Я. Програмування мовою С. Львів : Оріяна-Нова, 2006. С. 253-277.


## Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати завдання 1 згідно з варіантом.
3. Доповнити наведений нижче код функціями згідно з варіантом завдання 2.

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

```

```
typedef struct one_word
{
    char word[40];
    int count;
    struct one_word *next;
} pair;

typedef struct
{
    pair *head;
} word_list;

int push_back(word_list *L, char word[]);//додає елемент у список
int is_empty(word_list *L);//перевіряє, чи список пустий
void show_list(word_list *L);//виводить список на екран
//your code
int main()
{
    //your code
    return 0;
}
int push_back(word_list *L, char word[])
{
    pair *new_word =  (pair *)malloc(sizeof(pair));//виділення пам'яті під новий елемент
    if (new_word == NULL)
        return 1;//якщо пам'ять не виділено, повертається 1
    strcpy(new_word->word, word);
    new_word->count = 1;
    new_word->next = NULL;
    if (!is_empty(L)) //перевірка, чи список не пустий
    {
        pair *current = L->head;
        while (current->next != NULL)//пошук кінця списку
            current = current->next;
        current->next = new_word;//додавання нового слова у кінець
    }
    else L->head = new_word;//якщо пустий - новий елемент стає головою
    return 0;
}
void show_list(word_list *L)
{
    if (L->head == NULL)
```

```
47      {
48          puts("List is empty!");
49      }
50      pair *current = L->head;
51      while (current != NULL)
52      {
53          puts(current->word);
54          current = current->next;
55
56      }
57  }
58  int is_empty(word_list *L)
59  {
60      return L->head == NULL?1:0;
61  }
62  //your code
```

4. Оформити звіт до роботи.

## Варіанти завдань

### Варіант 1

1. Написати програму, яка приймає на вхід масив дійсних чисел та за допомогою динамічного стеку виводить цей масив у зворотному порядку.

2. Написати функції для роботи з лінійними списками:

- `insert_after` – вставляє елемент після заданого значенням або, якщо такого немає, додає у кінець;
- `remove_after` – видаляє один, кілька, або всі елементи списку після заданого.

### Варіант 2

1. Написати програму, яка приймає на вхід масив символів – арифметичний вираз, та за допомогою динамічного стеку перевіряє коректність розстановки дужок “`(),{},[]`”. Приклад виразу:

`y = 100*(25-{40*2+sin[2*pi*t]}).`

2. Написати функції для роботи з лінійними списками:

- `copy_list` – копіює один динамічний список в інший;
- `size_of_list` – рахує та повертає кількість елементів у списку.

### Варіант 3

1. Написати програму, яка створює чергу пацієнтів до лікаря. Обов’язкові функції: додати та видалити пацієнта, перевірити, чи пуста черга,

перемістити пацієнта у кінець черги, вивести чергу на екран. Поля структури даних: прізвище та рік народження.

2. Написати функції для роботи з лінійними списками:

- `split_list` – розділяє список на два за заданим роздільним елементом;
- `join_list` – об'єднує два динамічні списки в один.

**Варіант 4**

1. Написати програму, яка створює чергу клієнтів ЦНАП. Обов'язкові функції: додати та видалити клієнта, перевірити, чи пуста черга, вивести чергу на екран. Поля структури даних: прізвище, рік народження, послуга.

2. Написати функції для роботи з лінійними списками:

- `delete_list` – видаляє динамічний список;
- `reverse_list` – розвертає список від кінця до початку.

**Варіант 5**

1. Написати програму, яка створює чергу абітур'єнтів до приймальної комісії. Обов'язкові функції: додати та видалити абітур'єнта, перевірити, чи пуста черга, вивести чергу на екран, вивести чергу на екран за заданою спеціальністю. Поля структури даних: прізвище, спеціальність.

2. Написати функції для роботи з лінійними списками:

- `find` – знаходить заданий елемент та повертає його адресу, якщо не знайдено – `NULL`.
- `swap` – змінює місцями два елементи списку, задані своїми адресами.

## Запитання та завдання для самоперевірки

1. Що таке динамічні структури? З чого вони складаються?
2. Які види зберігання лінійних списків ви знаєте?
3. Які операції над лінійними списками можна виконувати?
4. Назвіть основні операції над стеком і поясніть їх виконання.
5. Як можна отримати доступ до будь-якого елемента стеку, крім вершини?
6. Поясніть, що таке черги, і які правила їх організації.
7. Назвіть основні операції над чергами і поясніть їх виконання.

# Список літератури


1. Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С. Житомир : ЖДТУ, 2007. 238 с.
2. Шпак З.Я. Програмування мовою С. Львів : Оріяна-Нова, 2006. 432 с.
3. Пінчук В.П., Лозовська Л.І. Програмування мовою С/С++ з прикладами та вправами : навчальний посібник. Запоріжжя : ЗНТУ, 2008. 197 с.
4. Сикорд Роберт С. Эффективный С. Профессиональное программирование. Санкт-Петербург : Питер, 2022. 304 с.
5. Алгоритми і структури даних: навчальний посібник / Т.О. Коротєєва. Львів : Видавництво Львівської політехніки, 2014. 280 с.
6. Керниган, Брайан У., Риттчи, Деннис М. Язык программирования С, 2-е изд. : пер. с англ. Москва : Издательский дом “Вильямс”, 2009. 304 с.
7. Хайнеман, Джордж, Пояяис, Гэри, Сеяков, Стэнли Алгоритмы. Справочник с примерами на С, С++, Java и Python, 2-е изд. : Пер. с англ. Санкт-Петербург : ООО “Альфа-книга”, 2017. 432 с.
8. Алгоритми і структури даних : посібник / Н.Б. Шаховська, Р.О. Голощук. Львів : Видавництво “Магнолія”, 2010. 216 с.
9. Структури даних та алгоритми : навч. посібник. / Михальов О.І., Крамаренко В.В., Ялова К.М., Новікова К.Ю. Дніпродзержинськ : ДДТУ, 2010. 286 с.
10. Програмування та алгоритмічні мови. Методичні вказівки до виконання лабораторних робіт для студентів напрямку підготовки 0908 ”Електроніка” по спеціальності 6.090803 “Електронні системи”. Частина 2. Мова програмування “С”/ Укл. Ревко А.С., Гордієнко В.В. –Чернігів: ЧДТУ, 2007. 60 с.

*Навчальне видання*

**Програмне забезпечення об’єктів кіберпростору. Мова програмування С**

Методичні рекомендації до лабораторних робіт

Укладачі : **Галюк** Сергій Дмитрович, **Круліковський** Олег Велерійович

Відповідальний за випуск **Шпатар Петро Михайлович**

Літературний редактор Колодій Оксана Віталіївна

Технічний редактор Кудрінська Оксана Михайлівна

Підписано до друку 29.09.2022. Формат 60х84/16.
Папір офсетний. Друк офсетний. Ум.-друк. арк. 5,6.
Обл.-вид. арк. 6,0. Зам. Н-106.
Друкарня видавництва Чернівецького національного університету
58002, Чернівці, вул. Коцюбинського,2

Свідоцтво про державну реєстрацію ДК №891 від 08.04.2002