КПІ ім. Ігоря Сікорського

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

ПРОГРАМУВАННЯ 1

Методичні вказівки
до

Комп'ютерного практикуму

Частина 1

"Програмування 1"

Київ 2020

Програмування, ч.1. **Програмування 1**: Метод. вказівки до комп'ютерного практикуму для студентів 1-го курсу
Уклад. О.І. Лісовиченко. – К.: КПІ ім. Ігоря Сікорського, 2020 – 48с.

Рекомендовано кафедрою
технічної кібернетики
ФІОТ
КПІ ім.Ігоря Сікорського
(Протокол №1 від 28.08.2020р.)

Навчальне видання

Програмування 1

МЕТОДИЧНІ ВКАЗІВКИ
ДО КОМП'ЮТЕРНОГО ПРАКТИКУМУ

Укладачі *Лісовиченко Олег Іванович*, к.т.н., доцент

Відповідальний редактор *Ткач Михайло Мартинович*,к.т.н, доцент

Рецензенти *Павлов Олександр Анатолійович*, д.т.н., професор

ЗМІСТ

Вступ

Комп‘ютерні практикуми з модуля “ Програмування ” виконуються студентами I курсу денної та заочної форми навчання.

Основна мета комп‘ютерного практикуму – закріпити знання, одержані на лекційних, практичних заняттях та в результаті самостійної роботи. Студенти опановують засоби опису даних та процедурні засоби мови С (ВС), розвивають навички складання програм та алгоритмізації задач, освоюють прийоми налагодження для одержання вірних результатів.

Мова програмування С є частиною програмного середовища Borland C, яке включає не лише відповідний компілятор, але і текстовий редактор, компоновник, завантажувач, налагоджувач та інші засоби, що прискорюють процес підготовки програми. Крім того, можливості мови С розширюються додатковими програмними засобами – модулями. Тому необхідно оволодіти прийомами роботи в середовищі Borland C.

В якості інструментів для навчання С використовується інтегроване середовище розробки Qt Creator 3.5.0, яке включає не лише відповідний компілятор, але і текстовий редактор, компонувальник, відлагоджувач, редактор інтерфейсу, та інші засоби, що прискорюють процес підготовки програми.

Фреймворк Qt 5, як один із засобів автоматизації програмування є крос-платформенним програмним засобом, що виконується у всіх популярних операційних системах (Windows, Linux, Unix, Mac OS X, iOS, Android, і т.д.).

При роботі в комп’ютерному класі інформаційних технологій необхідно вживати заходів для боротьби з програмними вірусами та дотримуватись правил роботи та техніки безпеки.

1. Виконання робіт

Підготовка до комп‘ютерного практикуму починається з вивчення теоретичного матеріалу, засвоєння основних положень, які перевіряються контрольними питаннями до кожної роботи.

Далі у відповідності до завдання треба зрозуміти й розробити алгоритм, саму програму та її логічну схему. Підготувати тестові приклади та перевірити їх в програмі. Якщо є потреба, усунути недоліки, та знову перевірити правильність виконання програми на тестових прикладах. Після цього необхідно підготовити звіт, де повинно бути текст та логічна схема програми. В схемі потрібно якомога менше використовувати дублювання коду програми та використовувати логічні словесні пояснення до алгоритму роботи програми.

На занятті студент показує звіт викладачеві, відповідає на поставлені питання щодо особливостей програмних засобів, типів даних та специфіки алгоритму.

При відсутності звіту або відповідних знань до роботи студент не допускається.

Перевірка програми здійснюється за даними які були завчасно підготовлені студентом, а також тестується прикладами, визначеними викладачем.

У разі виявлення помилок треба визначити їх причини та ліквідувати. Якщо помилки не очевидні, то потрібно використати засоби налагоджувача з контролем проміжних результатів та порівнянням їх із теоретичними розрахунками.

Перевірці підлягають всі можливі варіанти роботи програми. Після одержання вірних результатів, студент записує їх до звіту, робить висновки щодо правильності роботи програми і, за бажанням, дає рекомендації про можливі вдосконалення.

Завершений звіт студент має захистити під час співбесіди з викладачем, де повинен продемонструвати рівень своїх знань, самостійність та оригінальність своїх програмних рішень.

2. Вимоги до звіту

Звіт комп‘ютерного практикуму має містити:

1. тему роботи;
2. завдання, які необхідно виконати;
3. текст програми;
4. схему роботи програм;
5. введені та одержані результати;
6. перевірочні тестові приклади;
7. висновки щодо роботи програми.

Якщо робота складається з кількох вкладених задач, то по кожній з них реалізуються пункти 2-6.

Тема, завдання, деякі пояснення до виконання кожної роботи наводяться у методичних вказівках.

Якщо алгоритм задачі важко зрозуміти, то рекомендується спочатку скласти схему алгоритму, а далі переходити до реалізації програми.

Текст програми повинен легко сприйматись. Тому необхідно виділяти основні логічні частини і писати програму “сходинками”. На початку треба подавати коментарі про її призначення та автора-розробника. В тексті програми до її окремих частин також потрібно наводити пояснення. Оператор ***GOTO*** можна використовувати лише у ***виключних*** випадках.

Текст звіту виконується у відповідності до вимог стандартів [5] машинописним (на друкарській машинці) і машинним (на пристроях ЕОМ) способами. Формат аркуша - **А4.** Параметри сторінки (поля): верхнє – **25** мм, нижнє – **15** мм, ліве – **25** мм, праве – **10** мм. Основний текст набирається шрифтом **Times New Roman** розмір **12**, межрядковий інтервал - **1,0**. Для спеціальних символів і формул слід використовувати **Microsoft Equation** або **MathType Equation**. Наявність переносів не обов'язково. Всі малюнки, якщо вони створені засобами Microsoft Word, повинні бути згруповані в один об'єкт.

Дозволяється виконання звіту у рукописному вигляді.

До кожної з програм студент складає відповідну схему з урахуванням стандартів [4]. У додатку 1 наведено основні графічні символи схем, які необхідно використовувати при описі алгоритму роботи програми та правила їх виконання, приклад схеми подається у додатку 2. Виконувати схеми дозволяється вручну з дотриманням всіх вимог стандартів або на комп’ютері з використанням пакетів обробки текстової та графічної інформації:

Microsoft Word (вбудований модуль побудови схем доступний з панелі інструментів “Малювання”), Visio, OpenOffice, AutoCad, Компас та спеціалізованих програм мережі Internet.

У висновках до комп‘ютерного практикуму студент зазначає, чи збігаються одержані результати з теоретичними, що засвідчує правильність використаних засобів, відзначає труднощі, з якими він стикнувся під час налагодження та причини помилок.

Приклад звіту наведено у додатку 2.

3. Комп‘ютерні практикуми
3.1. Комп‘ютерний практикум № 1

3.1.1. Тема:

Структура програми мовою С. Освоєння прийомів роботи в середовищі ВС.

3.1.2. Теоретичні відомості

Програма на мові С може складатись з кількох функцій. Серед них обов’язково має бути функція з ім’ям main( ), з якої починається компонування.

В С опис змінних може розміщуватись в будь-якому місці програми перед звертанням до змінної. Властивості змінних залежать від того, де вони описані. Опис змінної реалізується таким чином:

тип ім’я_1, ім’я_2,…, ім’я_n;

При описі змінним дозволяється присвоювати початкові значення:

тип ім’я = значення;

Для підключення функцій стандартних бібліотек на етапі компонування до тексту програми треба включити заголовочні файли (header file) за допомогою директиви #include:

 # include <ім’я_файла>

Файли conio.h, stdio.h майже завжди повинні підключатись до програми. Файл conio.h містить описи функцій керування екраном в текстовому режимі. Наприклад в conio.h міститься опис функції getch(), яка зчитує символ і повертає його код; getch() може використовувати для затримки екрана DOS до натиснення будь-якої клавіші. Файл stdio.h містить функції вводу-виводу. Далі наведемо основні з них:

putchar(змінна) – виводить символ на екран;

printf(“керуючий_рядок”, змінна_1, змінна_2,…) – здійснює форматний вивід;

scanf(“керуючий_рядок”, &змінна_1, &змінна_2,…) – здійснює форматне введення

Керуючий рядок містить два типа інформації: символи, які безпосередньо виводяться на екран (для функції scanf – символи, які необхідно зчитати і відкинути) та команди формату (специфікатори формату), які визначають, як вводити та виводити аргументи. Специфікатори формату починаються символом % за яким іде код формату:

%c – символ;

%d – ціле десяткове число із знаком;

%i – ціле десяткове, вісімкове, шістнадцяткове;

%u – ціле беззнакове десяткове число;

%f – дійсне число з фіксованою крапкою;

%e – дійсне число з плаваючою крапкою;

%o – вісімкове число;

%x – шістнадцяткове число;

%s – рядок символів;

%p – покажчик;

%% - символ % (при виводі).

Кожному елементу списку виводу повинен відповідати специфікатор формату.

У списку виводу функції *printf* використовуються параметри значення, тому фактичними параметри можуть бути вирази.

Функція *scanf* повинна передати результати введення до головної програми, тому у списку вводу використовуються параметри адреси. Оскільки всі символи, які не є директивами перетворення, повинні з'явитись у потоці введення, то недоцільно зловживати зайвими символами у керуючому рядку.

Опис функцій здійснюється таким чином:

тип і'мя_функції (список параметрів)
{…
оператори;
…
return (вираз);
}

список параметрів включає *тип_параметра і'мя_параметра,…*

Оператор *return (вираз)* приводить до виходу з функції і може використовуватись для передачі значення функції. В тілі функції може бути кілька операторів return, а може не бути жодного. В цьому випадку повернення до викликаючої програми відбувається після виконання останнього оператора тіла функції. Тут *тип* визначає тип значення, яке передає функція за допомогою оператора return.

Необхідно пам'ятати, що коли виклик функції здійснюється перед її описом, то попередньо необхідно навести заголовок (прототип) функції:

тип ім'я_ функції (тип_параметра_1, тип_параметра_2, …, тип_параметра_n);

Оператор циклу:

while (умова) оператор;

оператор виконується до тих пір, поки *умова* – істина (не дорівнює 0). Замість *оператора* може використовуватись група операторів у фігурних дужках **{ }.**

3.1.3. Завдання:

Написати програму, яка переводить числа з арабської системи в римську.

Опис алгоритму

Для спрощення задачі вважати що кількість однакових символів в римській системі необмежена трьома (тобто 4 позначати IIII, 40 – XXXX і т.д.)

Написати функцію roman, параметрами якої є: *n* – число, яке необхідно перетворити; *a* – арабське число (береться з таблиці); *r* – римське число, що відповідає арабському *a* (див. таблицю). У функції організувати цикл, в якому виводити на екран символ *r*, після чого переприсвоїти *n = n – a*. Умова виходу з циклу – *n < a*. Функція повинна повертати значення n.

В основній програмі зчитати *n* – число, яке необхідно перетворити, і 7 разів викликати описану вище функцію (*n = roman (n, a, r*)). При цьому значення *n* буде змінюватись функцією, а значення *a* та *r* необхідно кожного разу змінювати згідно таблиці.

Таблиця відповідностей чисел

Арабська система	Римська система
1000	M
500	D
100	C
50	L
10	X
5	V
1	I

Текст програми

```
#include <stdio.h>;
#include <conio.h>;
int roman(int,int,char);
main()
{int a;
  printf("ENTER NUMBER\n");
  scanf(" %d",&a);
  a=roman(a,1000,'M');
  a=roman(a,500,'D');
  a=roman(a,100,'C');
  a=roman(a,50,'L');
  a=roman(a,10,'X');
  a=roman(a,5,'V');
  a=roman(a,1,'I');
  getch();
  return 0;
}
roman(int i,int j,char c)
{while (i>=j) {putchar(c);  i=i-j; }
  return(i);
}
```

3.1.4. Контрольні питання:

1. Яка структура програми на мові С? [2] с. 14-17, [3] с. 63-69.
2. Функція *printf* та її використання. [2] с. 149-151, [3] с. 69-72.
3. Функція *scanf* та її використання. [2] с. 152-155.
4. Що таке директиви препроцесора? [3] с. 63-69.

3.2. Комп‘ютерний практикум № 2

3.2.1. Тема:

Програмування лінійних алгоритмів.

3.2.2. Теоретичні відомості

До арифметичних операцій мови С відносяться:

– віднімання та унарний мінус (зміна знаку числа);

\+ додавання;

* множення;

/ ділення;

% остача від ділення (тільки для цілих);

++ збільшення на одиницю (increment);

– – зменшення на одиницю (decrement);

Операції додавання, віднімання, множення та ділення можуть застосовуватись до всіх вбудованих типів даних. Операції послідовного обчислення виконуються зліва направо, тобто спочатку обчислюється значення лівого операнда, потім вираз праворуч від знака операції. Якщо операнди мають один тип, то результат арифметичної операції матиме той же тип. Тому коли операція ділення / застосовується до цілих чи символьних змінних, залишок відкидається. Так 11/3=3, а 1/2=0.

Операція ділення по модулю % повертає залишок від ділення, може застосовуватись лише до цілих змінних.

Мова С дає користувачу ще дві дуже корисні операції, специфічні саме для мови С. Це унарні (з одним операндом) операції ++ та --. Операція ++ додає до операнда одиницю, операція – віднімає від операнда одиницю. Обидві операції можуть іти після або перед операндом (префіксна та постфіксна форма). Різниця у використанні префіксної ++x та постфіксної x++ форм полягає в тому, що:

x++ – значення змінної x спочатку використовується у виразі, а потім змінна збільшується на одиницю;

++x – змінна x спочатку збільшується на одиницю, а потім її значення використовується у виразі.

Старшість арифметичних операцій (1-ий - старший):

1	++, – –
2	– (унарний мінус)
3	*, /, %
4	+, –

Операції простого й складеного присвоювання виконуються з права на ліво. Для зміни порядку операцій використовуються круглі дужки.

3.2.3. Завдання:

Написати програму, яка за введеними сторонами трикутника обчислює його площу, периметр, висоти, бісектриси і медіани.

Опис алгоритму

Програма повинна зчитувати сторони трикутника і повертати його площу, периметр, а також всі висоти, бісектриси і медіани.

Формули для знаходження відповідних величин:

- площа: $S = \sqrt{p(p-a)(p-b)(p-c)}$;

- висота, проведена до сторони a: $h_a = \frac{2\sqrt{p(p-a)(p-b)(p-c)}}{a}$;

- медіана, проведена до сторони a: $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$;

- бісектриса, проведена до сторони a: $\beta_a = \frac{2}{b + c}\sqrt{bcp(p - a)}$,

де $p = \frac{a + b + c}{2}$ – півпериметр; a, b, c – сторони трикутника.

Для обчислення квадратного кореня можна скористатись бібліотечною функцією *sqrt(вираз)*. Для цього необхідно підключити файл ***math.h***.

Також бажано зробити функцію перевірки існування трикутника із заданими сторонами (сума довжин двох сторін трикутника більша довжини третьої сторони).

3.2.4. Контрольні питання:

1. Які особливості виконання арифметичних операцій в мові С? Специфічні операції ++ та --. [2] с. 52-54.
2. Скорочені форми типу *вираз1 операція = вираз2*. [2] с. 56-57.
3. Операція кома: *вираз1, вираз2*. [3] с. 41.
4. Оператор присвоєння і послідовність його виконання. [3] с.53-54, [2] с. 49-52.
5. Явне перетворення типів. [3] с. 42.

3.3. Комп‘ютерний практикум № 3

3.3.1. Тема:

Програмування розгалужених алгоритмів.

3.3.2. Теоретичні відомості

Програма може “думати” завдяки умовним операторам

Повна форма оператора if:

if (умова) оператор;

else оператор;

Якщо значення *умова* “істина”, то виконується *оператор* (ним може бути складений оператор – блок), наступний за умовою. Якщо умова хибна, то виконується оператор, наступний за ключовим словом *else*. Оператор *else* може бути відсутній, тоді якщо умова хибна, виконується одразу наступний оператор програми. В якості умови може бути довільний вираз, в операторі лише перевіряється чи являється значення виразу ненульовим (істинним) чи нульовим (хибним).

Вкладеним оператором if називається наступна конструкція:

if (x)

if (y) оператор1;

else оператор2;

В мові С оператор *else* асоціюється з найближчим *if* у відповідному блоці. В цьому прикладі *else* відноситься до оператора *if (y)*. Для того щоб віднести його до оператора *if (x)*, потрібно відповідним чином розставити операторні дужки:

if (x)

{ if (y) оператор1;}

else оператор2;

Оператор switch використовується для вибору одного з багатьох шляхів.

Switch(вираз)

{case константний_вираз: оператор;[break;]

…

default: оператор

}

Спочатку виконується *вираз* і його значення порівнюються із *константними_виразами*, зазначеними після оператора *case*. При співпаданні починають виконуватись *оператори*, що вказані після символу “ : ” і продовжують виконуватись поки не зустрінеться оператор *break* або кінець блока. Якщо співпадань немає, виконуються оператори, які йдуть після *default.*

3.3.3. Завдання:

3.3.3.1. Завдання 1:

Написати програму для обчислення числа $y = \sqrt[k]{x}$ із заданою точністю.

Опис алгоритму

За введеними x, k, ε (точність обчислень) програма повинна повертати значення $y = \sqrt[k]{x}$, де $k, x \in R$.

В програмі використати ітераційну формулу:

$$y_{i+1} = y_i + \frac{1}{k}\left(\frac{x}{y_i^{k-1}} - y_i\right)$$

Організувати цикл, в якому на кожному кроці обчислювати:

$$\delta = \frac{1}{k}\left(\frac{x}{y_i^{k-1}} - y_i\right),$$

тоді $y_{i+1} = y_i + \delta$. Умова виходу з циклу: $|\delta| < \varepsilon|$, де ε - наперед задана точність обчислень. Початкові присвоєння: δ=1, y=1.

Для обчислення степеня y_i^{k-1} використати оператор циклу.

При написанні програми слід урахувати всі можливі випадки некоректного введення даних. Зробити перевірку правильності введеного числа ***x*** в залежності від введеного значення ***k***.

3.3.3.2. Завдання 2:

Написати програму – календар, яка за введеною датою (виключно дня, місяця, року) виводить день тижня прописом.

Опис алгоритму

Розрахувати день тижня за формулою:

$$day = (\|365.25 * year\| + \|30.56 * month\| + date + n)\%7$$

де *year* – повний рік (4 цифри), *month* – порядковий номер місяця, *date* – день (число), *n* – поправка:

$$n = \begin{cases} 0, \text{ якщо } month > 2 \\ 1, \text{ якщо рік високосний і } month \le 2 \\ 2, \text{ якщо рік невисокосний і } month \le 2 \end{cases}$$

% – операція ділення по модулю (взяття остачі від ділення);

$\|a\|$ – ціла частина;

day вказує на день тижня (0 – Пн, 1 – Вт, …, 6 – Нд).

Рік буде високосним при виконанні любої із двох умов:

1. $year\ \%100 \neq 0$ і $year\ \%4 = 0$;

2. $year\ \%100 = 0$ і $year\ \%400 = 0$.

При написанні програми слід урахувати всі можливі випадки некоректного введення даних (дня, місяця, року).

Вивід організувати за допомогою оператора switch.

3.3.3.3. Завдання 3:

Написати програму для розв'язання кубічного рівняння $x^3 + ax^2 + bx + c = 0$, де *a, b, c* – коефіцієнти кубічного рівняння.

Опис алгоритму

Програма повинна зчитати коефіцієнти рівняння та повернути значення всіх коренів (дійсних і комплексних).

Підстановкою $x = y - \frac{a}{3}$ рівняння перетворюється на:

$$y^3 + py + q = 0.$$

Отже, необхідно обчислити:

$$\mathrm{p} = \mathrm{b} - \frac{\mathrm{a}^2}{3}; \quad q = \frac{2a^3}{27} - \frac{ab}{3} + c,$$

де *a, b, c* – коефіцієнти кубічного рівняння. Далі визначаємо

$$d = \frac{p^3}{27} + \frac{q^2}{4}.$$

Якщо **<u>d > 0</u>**, то рівняння має один дійсний корінь:

$$\mathrm{y}_1 = \mathrm{u} + \mathrm{v},$$

та два комплексно – спряжених:

$$\mathrm{y}_2 = -\frac{\mathrm{u}+\mathrm{v}}{2} + \mathrm{i}\cdot\frac{\sqrt{3}(\mathrm{u}-\mathrm{v})}{2};$$

$$\mathrm{y}_3 = -\frac{\mathrm{u}+\mathrm{v}}{2} - \mathrm{i}\cdot\frac{\sqrt{3}(\mathrm{u}-\mathrm{v})}{2},$$

де $u = \sqrt[3]{-\frac{q}{2} + \sqrt{d}}$, $\mathrm{v} = -\frac{\mathrm{p}}{3\cdot\mathrm{u}}$.

Якщо **d = 0**, то рівняння має три дійсних кореня, з них два однакових:

$$y_1 = \frac{3q}{p}\ ; y_2 = y_3 = -\frac{3q}{2p}\ .$$

Якщо **d < 0**, то рівняння має три різних дійсних кореня, які, на жаль, можна обчислити лише наближено:

$$y_1 = 2 \mid \sqrt[3]{r} \mid \cos\frac{\varphi}{3}\ ;$$

$$y_2 = 2 \mid \sqrt[3]{r} \mid \cos\frac{\varphi + 2\pi}{3};$$

$$y_3 = 2 \mid \sqrt[3]{r} \mid \cos\frac{\varphi + 4\pi}{3},$$

де $r = \sqrt{\frac{-p^3}{27}}$, $\varphi = \arccos\left(-\frac{q}{2r}\right)$.

Зробити перевірку правильності введених даних та перевірку правильності добутих коренів (лише для дійсних коренів), шляхом підстановки їх в рівняння. Для отримання значення π використати константу M_PI (***math.h***).

3.3.4. Контрольні питання:

1. Логічні вирази та відношення і особливості їх виконання. [3] с. 52-53.
2. Умовний оператор *if* та особливості його виконання. [1] с. 80-81, [2] с. 60-63.
3. Особливості виконання оператора *switch.* [2] с. 63-64.
4. Умовний вираз. [2] с. 57-58.

3.4. Комп‘ютерний практикум № 4

3.4.1. Тема:

Оператори циклу. Робота з масивами.

3.4.2. Теоретичні відомості

Для організації циклів існує два оператори *while* та *for*.

Перший оператор має вигляд:

while (умова) оператор;

Якщо *умова* не дорівнює нулю, то виконується *оператор*, який може бути як простим, так і складеним. Коли ж *умова* одразу дорівнює нулю, то *оператор* не виконується жодного разу.

Коли потрібно, щоб *оператор* виконувався принаймні один раз, то можна використати інший різновид:

do оператор while умова;

Загальний вигляд оператора *for*:

for (вираз1; вираз2; вираз3) оператор;

Ця інструкція for еквівалентна конструкції

```
вираз1;
while (вираз2) {
        оператор;
        вираз3;
}
```

Для реалізації циклу з параметром оператор використовується таким чином:

for (ініціалізація змінної; перевірка_умови; приростання змінної) оператор;

ініціалізація змінної використовується для присвоєння початкового значення параметру циклу; *перевірка_умови* – звичайно умовний вираз, який визначає умову продовження циклу; *приростання змінної* використовується для зміни параметра циклу кожного разу при повторенні циклу. Ці три частини повинні розділятись крапкою з комою. Виконання циклу продовжується поки умовний вираз є істиною. Це надає оператору *for* такі можливості:

- параметр циклу може бути не тільки цілим;
- крок зміни може бути довільним;
- в тілі циклу змінну можна додатково модифікувати.

Масивом називається сукупність даних одного й того ж типу, об'єднаних під одним ім'ям. Кожний елемент масиву визначається ім'ям масиву і порядковим номером елемента, який називається індексом. Індекс в мові С завжди ціле число. Необхідно пам'ятати, що нумерація елементів масиву завжди починається з 0.

Опис масиву в програмі:

тип <ім'я_масиву> [розмір1][розмір2]…[розмірN].

Найчастіше використовують одномірні масиви:

тип <ім'я_масиву> [розмір],

де *тип* – базовий тип елементів масиву, *розмір* – кількість елементів одномірного масиву.

Елементи багатомірних масивів розміщуються в пам'яті послідовно таким чином, що найпершим змінюється останній (правий) індекс.

Повне ім'я масиву: *float vect[30], mas[30];*

має значення адреси базового елементу

vect == &vect[0].

Воно є константою, яку не можна змінювати. Тому присвоювати масив масиву не можна, тобто: *vect = mas* - НЕПРИПУСТИМЕ.

3.4.3. Завдання:

3.4.3.1. Завдання 1:

Написати програму, яка повинна виводити таблицю значень синусів або косинусів (розрахованих за допомогою розкладання функції в ряд Тейлора) і табличних значень, а також їх різницю в заданому діапазоні із заданим кроком та точністю.

Опис алгоритму

Програма повинна зчитувати межі діапазону [x_1, x_2] і крок зміни аргументу *dx* (в градусах), а також точність обчислень ε та виводити на екран таблицю з чотирма стовпчиками: значення аргументу *x* (в градусах); значення $sin(x)$ або $cos(x)$, розраховане за допомогою розкладання в ряд Тейлора; значення $sin(x)$ ($cos(x)$), розраховане стандартними функціями $sin(x)$ ($cos(x)$) різницю між табличними та розрахованими значеннями.

Необхідно пам'ятати, що при обчисленні значення синуса в обох випадках необхідно перевести значення аргументу в радіани ($\text{радіан}=\frac{\text{градус} * \pi}{180}$).

Розкладання функцій в ряд Тейлора:

$$sin\,x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots; \qquad cos\,x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

В програмі необхідно організувати вкладений цикл. У зовнішньому циклі на кожному кроці змінювати поточне значення аргументу $x = x + dx$ (початкове присвоєння $x = x_1$), обчислювати значення $sin(x)$ або $cos(x)$ стандартною функцією (для порівняння з результатами, одержаними за допомогою розкладання функції в ряд) і організувати вивід рядків таблиці на екран. Умова виходу із зовнішнього циклу $x = x_2$. У внутрішньому циклі розраховувати значення $sin(x)$ за допомогою розкладання в ряд Тейлора: необхідно обчислювати на кожному кроці $\delta = \delta * \frac{-x^2}{(n+1)(n+2)}$, $sin(x) = sin(x) + \delta$, $n = n + 2$. Початкові присвоєння: $\delta = sin(x) = 1$ (x – поточне значення аргументу, що обчислюється в зовнішньому циклі), $n = 1$.

У внутрішньому циклі розраховувати значення $cos(x)$ за допомогою розкладання в ряд Тейлора: необхідно обчислювати на кожному кроці $\delta = \delta * \frac{-x^2}{2*n*(2*n-1)}$, $cos(x) = cos(x) + \delta$, $n = n + 1$. Початкові присвоєння: $\delta = cos(x) = 1$ (x – поточне значення аргументу, що обчислюється в зовнішньому циклі), $n = 1$. Умова виходу із внутрішнього циклу $|\delta| < \varepsilon$, де ε – наперед задана точність обчислень. Після виходу із внутрішнього циклу $sin(x)$ або $cos(x)$ прийме шукане значення для поточного значення аргументу *x*.

3.4.3.2. Завдання 2:

Написати програму для впорядкування (за зростанням та спаданням) масиву дійсних чисел.

Опис алгоритму

Програма повинна зчитувати кількість елементів масиву *n* і сам масив дійсних чисел та виводити на екран впорядкований (по зростанню або спаданню, за вибором користувача) масив.

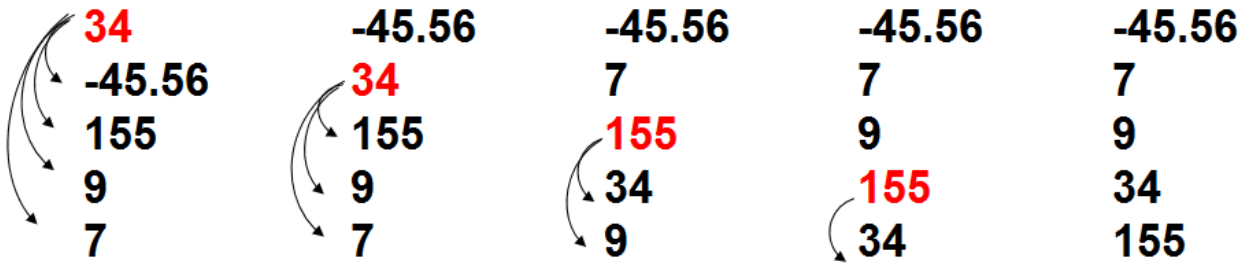
Алгоритм "бульбашки".

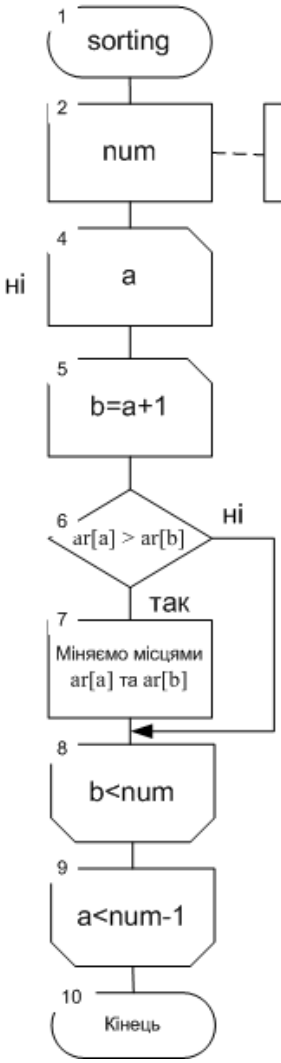
Цей алгоритм простий для реалізації але найменш ефективний. В ньому використовується вкладений цикл. Кількість кроків в зовнішньому циклі на одиницю менша кількості елементів масиву ($1 \le i \le n-1$), у внутрішньому циклі дорівнює $n - i$ (i – номер ітерації зовнішнього циклу), тобто $1 \le j \le (n-i)-1$. У внутрішньому циклі на *j*-му кроці *j*-1–ий елемент порівнюється з *j*–им, і якщо він більший, елементи міняються місцями. Таким чином після першого проходження внутрішнього циклу найбільший елемент переміститься в кінець масиву.

Наприклад, є послідовність елементів масиву: 34 -45.56 155 9 7.

Після сортування за алгоритмом по зростанню, послідовність елементів масиву має вигляд: -45.56 7 9 34 155.

Наведемо наступні етапи сортування:






```
// сортування за зростанням
void sort(float *ar,int num)
{ float t;
  for(a=0;a<num-1;a++)
   for(b=a+1;b<num;b++)
   {  if(ar[a]>ar[b])
      {  t=ar[a];
         ar[a]=ar[b];
         ar[b]=t;
       }
    }
 }
```

3.4.4. Контрольні питання:

1. Оператор *while (do – while).* [3] с. 90-91.
2. Які особливості виконання оператора *for*? [3] с. 91-96.
3. Що таке масив? Як описуються масиви? [3] с. 101-109.
4. Оператори *break, continue.* [3] с. 96-101.

3.5. Комп‘ютерний практикум № 5

3.5.1. Тема:

Покажчики і масиви. Робота з рядками.

3.5.2. Теоретичні відомості

Покажчик – це змінна, яка містить адресу деякого об’єкта (її значення – деяке ціле число). Покажчики описуються таким чином:

*Тип *<ім’я_змінної>;*

тут *тип* визначає тип об’єкта, на який вказує покажчик.

З покажчиками пов’язані дві спеціальні унарні (тобто мають один операнд) операції: & та *. Операція & відповідає операції “взяти адресу”. Операція * відповідає словам “значення, що знаходиться за даною адресою”. Також над покажчиками можна виконувати деякі арифметичні операції:

1. додати або відняти від покажчика цілу константу (при цьому константа означає не кількість байт, а кількість одиниць даних, тобто $p = p \pm n$ еквівалентно $< p > \; = \; < p > \pm \, n \, * < \textit{кількість байт пам'яті базового типу} >$;

2. віднімання покажчиків один від одного.

Інші арифметичні операції над покажчиками ЗАБОРОНЕНІ. Покажчики можна також порівнювати (дозволені всі 6 операцій: $<$, $>$, $\leq$, $\geq$, $==$, $!=$). До покажчиків можна застосовувати операцію присвоєння. Покажчики одного типа можна використовувати в операціях присвоєння як звичайні змінні. Покажчику на void (порожній тип) можна присвоїти покажчик будь-якого іншого типу, однак при зворотному присвоєнні потрібно явно перетворювати тип. На відміну від повного імені масиву покажчик є змінною.

В мові С рядок – це масив символів, в кінці якого знаходиться нульовий байт, який позначається символьною константою ‘\0’. В мові С немає стандартного типу рядка і він описується як масив символів. Для роботи з масивом символів як з рядком є набір бібліотечних функцій.

Для вводу рядка можна скористатись функцією *scanf*() зі специфікатором формату *%s* (слід пам’ятати, що функція *scanf*() вводить символи до *першого символу пропуску*), або спеціальною бібліотечною функцією gets(), описаною в файлі ***stdio.h*** (функція *gets*() дозволяє вводити рядки, що містять символи пропусків, ввід завершується натисненням клавіші *Enter*). Обидві функції автоматично ставлять в кінець рядка нульовий байт (не забудьте зарезервувати для нього місце). Як параметр в обох цих функціях використовується просто ім’я масиву.

Вивід рядків реалізується функціями *printf*( ) зі специфікатором формата *%s* та бібліотечною функцією *puts*( ), описаною в файлі ***stdio.h***. Обидві функції виводять вміст масиву до першого нульового байта. Функція *puts*( ) додає в кінці рядка, який виводиться, символ переходу на новий рядок.

Для порівняння рядків (в лексикографічному розумінні) існує бібліотечна функція

strcmp(рядок1, рядок2),

опис якої знаходиться в файлі ***string.h***. Ця функція порівнює два рядки і якщо перший більше, повертає значення більше нуля, якщо рядки однакові – повертає 0, якщо перший рядок менше другого – повертає значення менше нуля.

3.5.3. Завдання:

3.5.3.1. Завдання 1:

Написати програму для впорядкування масиву рядків, використовуючи покажчики.

Опис алгоритму

Програма повинна зчитати кількість елементів масиву рядків і сам масив та вивести на екран впорядкований масив рядків.

Для реалізації програми потрібно написати функцію впорядкування масиву рядків, параметрами якої є масив покажчиків на char та кількість елементів масиву. Звертатись до рядків через їх адреси, записані в масив покажчиків.

В програмі потрібно створити двомірний символьний масив, в який записати введені рядки, і масив покажчиків на *char*, в який записати адреси рядків, і потім використовувати як фактичний параметр при виклику процедури впорядкування.

```
char  list[n_str][str_size];
char *addr[n_str];
int i,n;
…
for (i=0; i<n; i++)  addr[i]=list[i];
…
```

Тут *n* – кількість елементів масиву рядків *list*. В даному випадку *list[i]* має значення адреси *i*-ого рядка (адреси першого елемента рядка).

3.5.4. Контрольні питання:

1. Що таке покажчик? Які операції можна виконувати над покажчиками? [2] с. 94-96.
2. Як організована робота з рядками в С? [2] с. 104-107.
3. Покажчики та масиви: спільні та відмінні властивості. [2] с. 98-100.

3.6. Комп‘ютерний практикум № 6

3.6.1. Тема:

Використання динамічних масивів.

3.6.2. Теоретичні відомості

У разі обмежень на обсяг статичної пам’яті можна використати динамічну, яка знаходиться не у сегменті даних, а в купі (*heap*).

Для цього передбачені функції:

*void *malloc (unsigned n),*

яка повертає покажчик на початок області динамічної пам’яті розміром *n* байт.

*void *calloc (unsigned n, unsigned m),*

яка повертає покажчик на початок блоку динамічної пам’яті для розміщення *n* елементів по *m* байт кожний. На відміну від попередньої функції *виділена область обнуляється*.

Якщо потрібну пам’ять виділити не можна, функції передають **NULL**.

Функція

void realloc (*void *dump, unsigned ns*)

змінює розмір блоку раніше виділеної динамічної пам’яті з початковою адресою *dump* до розміру *ns* байтів. Якщо пам’ять не виділялась ($dump == NULL$), то функція виконується як *malloc.*

Зазначені функції передають безтиповий покажчик. Для даних певного типу треба явно зазначити відповідний тип покажчика. Наприклад, масив подвійних дійсних з *n* елементів можна задати покажчиком *double *m.* Тоді виділення динамічної пам’яті для цього масиву матиме вигляд:

*m = (double *) calloc (n, sizeof (double)).* (1)

Звільняє пам’ять функція

*free (void *a).*

3.6.3. Завдання:

Написати програму розв’язання системи лінійних алгебраїчних рівнянь (СЛАР) методом простої ітерації з використанням динамічних масивів.

Опис алгоритму

СЛАР має вигляд:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_1 \end{cases} \quad (2)$$

або у векторній формі:

$$AX = B.$$

З кожного рівняння визначимо x_i ($i = 1..n$) і одержимо:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}\left(b_1 - a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n\right) \\ x_2 &= \frac{1}{a_{22}}\left(b_2 - a_{21}x_1 - a_{23}x_3 - \ldots - a_{2n}x_n\right) \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n &= \frac{1}{a_{nn}}\left(b_n - a_{n1}x_1 - a_{n2}x_2 - \ldots - a_{nn}x_n\right) \end{aligned} \qquad (3)$$

Або у векторній формі одержали співвідношення:

$$\overline{X} = F(\overline{X}),$$

для якого можна застосувати метод простої ітерації. Метод полягає в тому, що у праву частину підставляємо якісь значення x_i ($i = 1..n$), одержуємо нові значення, які знову підставляємо у праву частину і т.д. Отже, на кожній ітерації маємо попередні значення невідомих xp_i ($i = 1..n$), які підставляємо до правої частини схеми (3) і нові значення x_i ($i = 1..n$), які одержуємо в лівій частині.

Обчислювальну схему (3) можна записати:

$$x[i] = (b[i] - \sum_{\substack{j=1 \\ j \neq i}}^{n} (a[i][j] \cdot xp[j])) / a[i][i], \quad i = 1..n \qquad (4)$$

Початкові значення невідомих можна обчислити:

$$xp[i] = b[i] / a[i][i], \; i = 1..n \qquad (5)$$

Після кожної ітерації значення певного невідомого змінюється на величину

$delta_i = | x[i] - xp[i] |$, де $|a|$ – абсолютна величина.

Обчислення можна припинити, коли максимальна поправка

$$prop = \max_i (delta_i)$$

стане менше наперед заданої точності eps.

По закінченні ітерації, обчислення всіх нових значень x_i, визначення максимальної поправки *prop* треба переписати масив *x* до масиву *xp*.

Необхідною (але не достатньою) умовою збіжності методу є виконання наступної умови:

$$| a_{ii} | > \left| \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} \right|.$$

Матрицю коефіцієнтів рівняння *A*, вектори *B, X, XP* реалізуються як динамічні масиви. Оскільки одновимірний масив можна задати покажчиком

d*ouble *b,*x,*xp,*

то двомірний масив можна задати покажчиком на одномірні масиви, або покажчиком на покажчик

*double **a;*

Виділення пам'яті під одномірні масиви здійснюється відповідно до (1).

Певні особливості має виділення пам'яті під двомірний масив. Спочатку у динамічній пам'яті треба створити масив покажчиків на рядки матриці:

*a = (double **) calloc (n, sizeof (double *));*

Потім через масив покажчиків *a[i]* треба виділити пам'ять для кожного рядка матриці:

for (i = 0; i < n; i++)

*a[i] = (double *) calloc (n, sizeof (double));*

Після цього до елементу матриці можна звертатись за звичайним ім'ям *a[i][i]*.

Звільнення пам'яті здійснюється у зворотному порядку: спочатку звільняємо масив рядків

for (i = 0; i < n; i++)

free (a[i]);

а потім пам'ять під масив покажчиків

free (a);

3.6.4. Контрольні питання:

1. Покажчики та дії з ними.
2. Відмінність покажчиків у мові С та у мові Turbo Pascal.
3. Покажчики та масиви. [3] с.184-191.
4. Засоби роботи з динамічною пам'яттю. [3] с.179-184.

3.7. Комп‘ютерний практикум № 7

3.7.1. Тема:

Функції та покажчики на функції.

3.7.2. Теоретичні відомості

Ім’я функції має значення адреси, починаючи з якої вона розміщується у пам’яті і, певною мірою, є константою.

У деяких випадках треба мати аналог функції – змінної, наприклад, в залежності від певних умов обчислювати різні функції, або написати функцію інтегрування певного класу функцій. Тобто, потрібна функціональна змінна, яка реалізується за допомогою покажчика на функцію.

Якщо є кілька функцій певної структури, наприклад:

double fun1 (double, int);

double fun2 (double, int);

double fun3 (double, int);

то покажчик на такі функції матиме вигляд

*double (*fpr) (double, int);*

Тут дужки *(*fpr)* потрібні, бо модифікатор ( ) має перевагу над модифікатором *. Тому:

*double *fpr (double, int);*

означатиме функцію, яка передає покажчик на *double.*

За допомогою покажчиків на функцію, можна створювати масиви функцій, використовувати їх як параметри інших функцій.

Наприклад, для числового інтегрування зазначених вище функцій можна створити універсальну функцію, яка має параметрами не лише межі інтегрування, точність, але і функцію, для якої обчислюється інтеграл

*double integrate (double (*fpr)(double), double lowbow, double highbow, double eps);*

При виклику такої функції замість формального параметру – покажчика на функцію треба зазначити ім’я фактичної функції.

s = integrate (fun2, a,b,eps);

3.7.3. Завдання:

Написати програму для обчислення коренів нелінійних рівнянь на заданому проміжку.

$$\cos\frac{t}{x} - 2\sin\frac{1}{x} + \frac{1}{x} = 0, x \in [a1;a2];$$

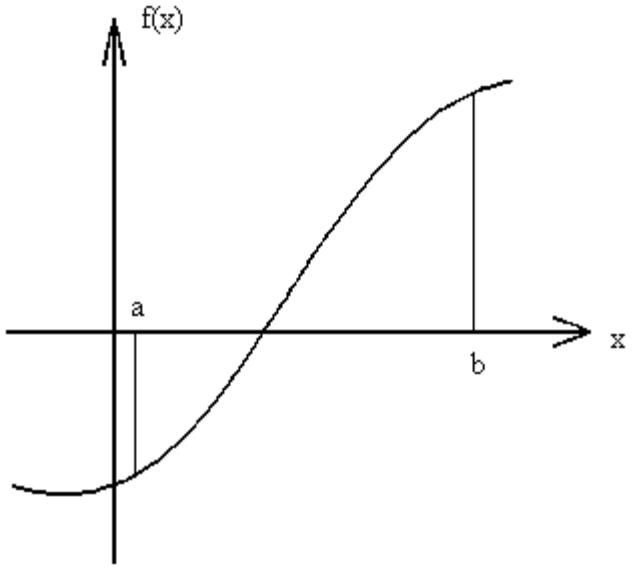
$$\sin(\ln x) - \cos(\ln x) + t\ln x = 0, x \in [a1;a2]$$

, де $t, a1, a2 \in R$.

Реалізувати обчислення 2-х функцій за вибором користувача та методу розрахунку, теж за вибором користувача.

Опис алгоритму

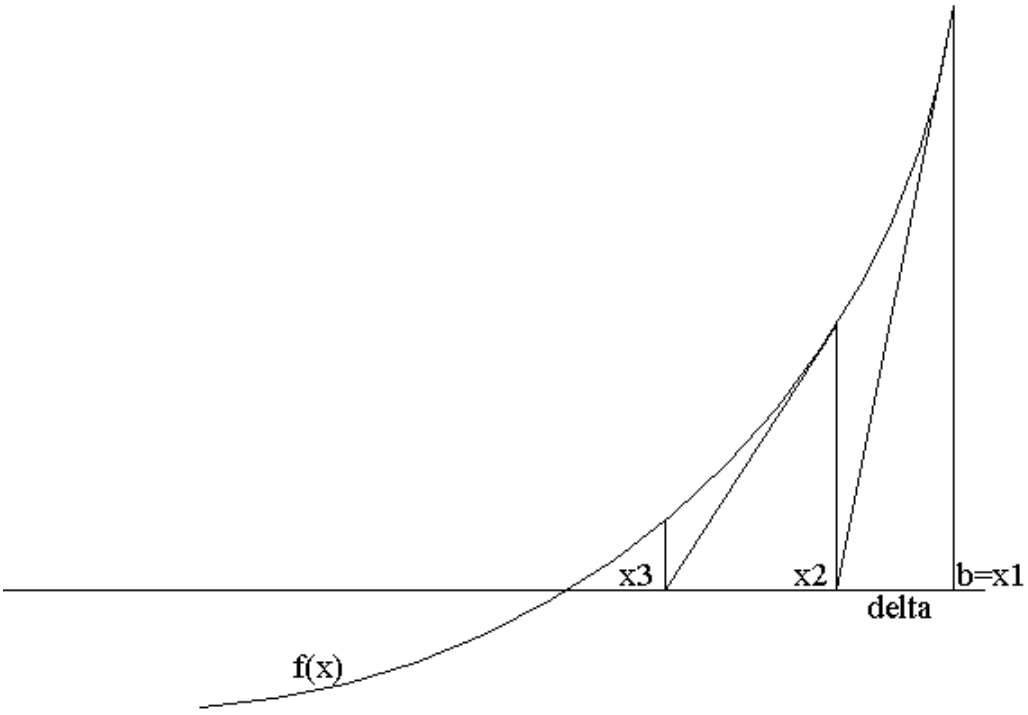
Корені можна обчислити або методом половинного ділення або методом дотичних (Ньютона).



Метод ***половинного ділення*** дозволяє знайти один корінь на заданому інтервалі.

На кожному кроці обчислюється значення $x = \frac{a+b}{2}$ і перевіряється знак добутку $f(a) \cdot f(x)$. Якщо $f(a) \cdot f(x) > 0$ то відбувається присвоєння $a = x$, в протилежному випадку $b = x$. Ці дії повторюються поки $|b - a| > \varepsilon$, де ε – наперед задана точність.

Метод ***Ньютона*** полягає в наступному:



1. Знаходиться значення функції в крайній точці – *f(b)*;
2. Через знайдену точку проводимо дотичну. Знаходимо точку x2. Алгебраїчно це виглядає наступним чином:

$$delta = \frac{f(x_i)}{f'(x_i)}; \quad x_{i+1} = x_i - delta;$$

де $f'(x_i)$ - похідна функції в точці x_i.

3. Повторюємо п.2 доти, поки абсолютна величина поправки *delta* більша за наперед задану точність *eps*.

Для знаходження похідної можна скористатись формулою:

$$f'(x_i) = \frac{f(x_i + \alpha) - f(x_i)}{\alpha},$$

якщо α – додатнє, близьке до нуля число (підбирається).

В програмі необхідно написати дві функції для обчислення значення правої частини кожного рівняння і функцію для знаходження кореня рівняння на інтервалі, одним з параметрів якої зробити покажчик на функцію і передавати до неї необхідну функцію для розв'язання відповідного рівняння.

3.7.4. Контрольні питання:

1. Структура функції. [2] с. 71-76, [3] с. 110-113, с. 203-206.
2. Прототипи функцій. [3] с. 206-208.
3. Способи передачі параметрів: параметри – значення, параметри – адреси. [2] с.96-98, [3] с. 207-213.
4. Відмінності функції мови С від інших мов.
5. Складні імена з модифікаторами *, ( ), [ ]. [2] с. 119-122.
6. Покажчики на функції та їх використання. [2] с. 117-119, [3] с. 227-238.

3.8. Комп‘ютерний практикум № 8

3.8.1. Тема:

Структури.

3.8.2. Теоретичні відомості

Структура – це одна або кілька змінних (можливо різного типу), які для зручності роботи з ними згруповані під одним іменем. Структури допомагають в організації складених даних (особливо у великих програмах), оскільки дозволяють групу зв’язаних між собою змінних розглядати не як множину окремих елементів, а як єдине ціле.

Опис структури починається з ключового слова struct і містить список описів у фігурних дужках. Після слова struct може слідувати ім’я, яке називається *тегом* (від англ. слова *tag* – ярлик, етикетка) або шаблоном структури. Шаблон дає назву структурі даного виду і далі може слугувати коротким позначенням тієї частини опису, яка міститься в дужках.

struct ім’я

{

опис елементів

} [змінна1, змінна2];

Перелічені в дужках змінні називаються елементами структури.

Елементами структур можуть бути також масиви, структури, масиви структур.

При описі структури ніякі змінні не створюються і пам’ять під них не виділяється. Опис структури визначає тип. Для опису змінних даного типу необхідно написати:

struct ім’я змінна1, змінна2,…;

або відразу при описі структури вказати список змінних:

struct ім’я

{

описи

} список_змінних;

Доступ до конкретного елемента структури здійснюється за допомогою операції “.” (“крапка”) :

ім’я_змінної . ім’я_поля;

Якщо описані дві змінні типу структури з одним шаблоном, то над ними можна виконати операцію присвоєння:

struct ім’я змінна1, змінна2;

змінна1= змінна2;

При цьому відбудеться побітове копіювання кожного елементу другої змінної у відповідний елемент першої. Не можна використовувати операцію присвоєння для змінних типу структури, шаблони яких описані під різними іменами, навіть якщо вони описані ідентично.

Структури можуть використовуватись як параметри в процедурах.

Можна описати покажчик на структуру і передавати аргумент типу структури за посиланням.

*struct ім'я *покажчик;*

Якщо передавати структуру за значенням, то всі елементи структури заносяться до стеку. Якщо, наприклад, елементом структури є масив, то стек може переповнитись. При передачі за посиланням до стеку заноситься тільки адреса структури. При цьому копіювання структури не відбувається, також з'являється можливість змінювати вміст елементів структури.

Використання покажчиків на структуру зустрічається часто. Тому для доступу до окремого елементу структури крім способу *(*покажчик_на_структуру).ім'я_поля* є ще один. В мові С вводиться спеціальна операція –> ("стрілка"). Операція "стрілка" застосовується замість операції "крапка", коли необхідно використовувати значення елемента структури із застосуванням змінної покажчика:

покажчик_на_структуру –> ім'я_поля;

Мова С має засіб, який дозволяє давати типам нові імена:

typedef тип нове_ім'я_типу;

Нове ім'я можна застосовувати в описах, операторах явного перетворення типів і т.д. так само як і стандартне.

Наприклад, якщо необхідно в програмі замінити тип float на REAL:

typedef float REAL;//замінити тип ***float*** *ім'ям* ***REAL***

Далі в програмі можна писати: *REAL a, b, c;*

Фактично нових типів тут не створюється, а надається нова назва існуючим типам.

Ясно, що те ж саме можна зробити за допомогою директиви препроцесора:

*#define REAL float; /*REAL скрізь буде замінене на float*/*

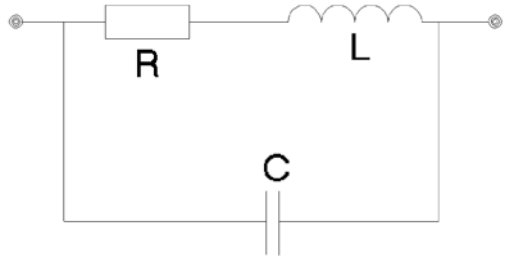
Однак існують і певні відмінності:

1. визначення ***typedef*** виконується компілятором, а не препроцесором;
2. функція typedef надає назву тільки типу даних і не може надавати ім'я константі;
3. функція typedef має більш широкі можливості чим пре процесор.

3.8.3. Завдання:

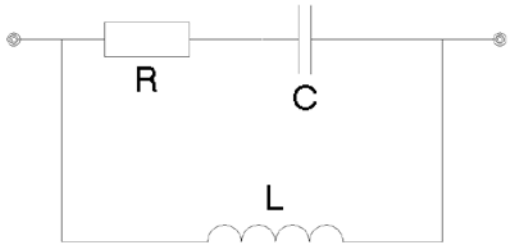
Написати програму для обчислення комплексного опору заданого коливального контуру в залежності від частоти струму. Варіанти коливальних контурів наведені нижче:

а)



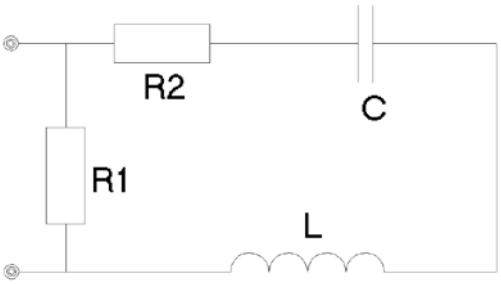
$$Z = \frac{\frac{L}{C} - i\frac{R}{\omega C}}{R + i(\omega L - \frac{1}{\omega C})}$$

б)



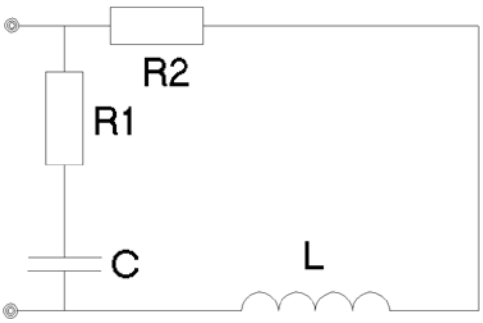
$$Z = \frac{\frac{L}{C} + i\frac{R}{\omega C}}{R + i(\omega L - \frac{1}{\omega C})}$$

в)



$$Z = \frac{R_1 R_2 + iR_1(\omega L - \frac{1}{\omega C})}{R_1 + R_2 + i(\omega L - \frac{1}{\omega C})}$$

г)



$$Z = \frac{R_1 R_2 + \frac{L}{C} + i(\omega L R_1 - \frac{R_2}{\omega C})}{R_1 + R_2 + i(\omega L - \frac{1}{\omega C})}$$

Тут $i = \sqrt{-1}$, кутова частота ω=2πf.

За вказівкою викладача визначається контур та його параметри R_1, R_2(Ом), L(мГн), C(мкФ).

Опис алгоритму

Програма повинна зчитати максимальне f_{max} і мінімальне f_{min} значення частот і крок *df* зміни частоти та вивести на екран таблицю значень поточної частоти *f*, та комплексного опору *Z*, а також значення резонансної частоти f_0.

Для реалізації програми потрібно за допомогою оператора *typedef* описати тип complex як структуру з двома елементами типу float. Необхідно написати функції для ділення двох комплексних чисел і для виводу комплексного числа на екран, параметрами яких будуть змінні описаного раніше типу complex.

Як варіанти можливе використання класів або перевантаження операцій.

Значення резонансної частоти обчислюється за формулою:

$$f_0 = \frac{1}{2\pi\sqrt{Lc}}$$

В програмі потрібно організувати цикл по *f* ($f_{min} \le f \le f_{max}$), в якому обчислювати значення ω і значення *Z* і виводити значення *f* та *Z*. Розрахунок проводити **БЕЗ** перетворення введених значень (~~мГн→Гн~~, ~~мкФ→Ф~~).

3.8.4. Контрольні питання:

1. Структури, їх опис, імена елементів структури. [2] с. 125-127, [3] с. 275-277.
2. Масиви структур. [2] с. 130-133. [3] с. 292-295.
3. Функції та структури. [2] с. 127-130.
4. Покажчики на структури. [2] с.133-135, [3] с. 295-296.
5. Поля. [2] с. 145-147, [3] с. 320-325.
6. Об’єднання. [2] с. 143-145, [3] с. 315-316.
7. Директива typedef. [2] с. 142-143.
8. Особливості структур як класів у C. [6] с. 281-290.
9. Перевантаження операцій у C. [6] с. 227-230.

3.9. Комп‘ютерний практикум № 9

3.9.1. Тема:

Робота з файлами.

3.9.2. Теоретичні відомості

В мові С файл – це послідовність байтів, кількість яких не визначена. Внутрішня структура файлу не задається. Робота з файлами здійснюється засобами операційної системи, файл не є типом даних.

Обробка файлів здійснюється за допомогою функцій, які можна розділити на такі групи:

1. Потокового обміну;
2. Консольного обміну;
3. Нижнього рівня;
4. Безпосереднє використання засобів операційної системи.

Розглянемо деякі особливості обробки файлів на рівні потоків. Такий обмін є буферизованим і може бути форматизованим.

Файл у програмі визначається певним покажчиком

*FILE *ім’я_покажчика;*

Цей покажчик посилається на структуру, яка містить інформацію про файл (адреса буфера, положення поточного символу в буфері, відкритий файл для читання або запису, чи були помилки при роботі з файлом, а також, чи не зустрівся кінець файлу). Ця структура має ім’я *FILE*, її опис знаходиться в ***stdio.h***.

Зазначимо, що *FILE* – це ім’я типу, а не шаблон структури. Воно визначено за допомогою *typedef*.

Для того щоб можна було читати з файлу або писати до файлу він попередньо повинен бути відкритий за допомогою бібліотечної функції *fopen*, яка встановлює зв’язок програмного файлу з фізичним. Функція повертає покажчик на файл:

ім’я_покажчика = fopen (зовнішнє_ім’я_файла, режим);

зовнішнє_ім’я_файла – рядок, який містить системне ім’я файлу, *режим* – рядок, в якому вказується яким чином буде використовуватись файл. Можливі наступні режими: “r” (read) – читання , “w” (write) – запис, “a” (append) – доповнення та ін.

Якщо неіснуючий файл відкривається для запису або доповнення, то він створюється (якщо така процедура фізично можлива). Відкриття існуючого файлу для запису призводить до втрати існуючої в ньому інформації. В разі появи помилки при відкритті файлу *fopen* повертає NULL.

Форматний ввід-вивід до файлів реалізується за допомогою функцій *fprintf* та *fscanf*. Вони ідентичні *printf* та *scanf* з тією різницею, що першим аргументом в них є покажчик файлу:

fprintf (покажчик_файла, форматний_рядок, ...);

fscanf (покажчик_файла, форматний_рядок, ...);

Функція

fclose (покажчик_файла);

розриває зв'язок між файловим покажчиком і файлом (який раніше був встановлений функцією *fopen*), звільнюючи цей покажчик і буфер для інших файлів.

Для реалізації прямого доступу до файлів в мові С існує можливість позиціювання внутрішнього покажчика (який вказує на поточний байт у файлі).

Функції

ipos = ftell (покажчик_файла);

fgetpos (покажчик_файла, &ipos);

– повертають поточне значення покажчика в змінну *ipos*;

fsetpos (покажчик_файла, ipos) – встановлює покажчик в позицію *ipos*;

rewind (покажчик_файла) – встановлює покажчик на початок файлу;

fseek (покажчик_файла, кількість_байт, start) – переміщує покажчик на необхідну кількість байт. Якщо *кількість_байт* > 0, покажчик переміщується вперед, якщо *кількість_байт* < 0 – назад. s*tart* – константа, яка вказує відносно чого треба здійснити переміщення:

0 або SEEK_SET – від початку файлу;

1 або SEEK_CUR – від поточного покажчика;

2 або SEEK_END – від кінця файлу.

Приклад:

```
FILE *fil;
struct rec{char name[40]; float square; int nas;};
...
fseek (fil, 0L, SEEK_SET); //на початок файлу
fseek (fil, n, SEEK_CUR); //на n байт вперед від поточної позиції
fseek (fil, -n, SEEK_CUR); //на n байт назад від поточної позиції
fseek (fil, 0L, SEEK_END); //на кінець файлу
```

3.9.3. Завдання:

Написати програму, яка виконує наступні дії:

1. Створення файлу;
2. Відкриття вже створеного файлу та завантаження даних з файлу;
3. Запис в файл даних (наприклад: назва області, площа, кількість населення), введених з клавіатури користувачем;
4. Видалення запису(ів) із файлу;
5. Видалення файлу;
6. Редагування запису(ів) із файлу;
7. Впорядкування (по вибору користувача, за зростанням або спаданням) записів в файлі за полями: назва області, площа, кількість населення;
8. Вставка у впорядкований файл записів так, щоб файл залишився впорядкованим;
9. Зчитування записів із файлу і виведення їх на екран.

Опис алгоритму

В програмі необхідно описати за допомогою оператора *typedef* тип *record* як структуру з полем типу **char* і двома полями типу *float*. Потрібно написати функції сортування (алгоритм сортування довільний) записів файла. З файлами необхідно працювати як з файлами прямого доступу. В програмі зчитувати кількість записів і дані. А також організувати перевірку корректності виконання функцій (перевірки на некоректне введення даних, сортування пустого чи неіснуючого файлу тощо). Інтерфейс вибору дії реалізувати у вигляді меню з можливістю доповнення функцій.

3.9.4. Контрольні питання:

1. Файли та їх особливості у мові С. [2] с. 155-158, [3] с. 325-368.
2. Чотири рівня функцій обробки файлів та їх відмінність.
3. Визначення файлу як потоку. Стандартні потоки.
4. Особливості обміну з файлами на нижньому рівні. Дескриптор файлу. [2] с. 164-169, [3] с. 369-381.
5. Послідовний та прямий доступ до файлів.
6. Текстові файли та їх особливості.

Література

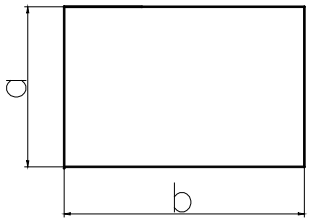
1. Юлин В. А., Булатова И. Р. Приглашение к СИ. – Минск: Вышэйша школа, 1990. – 224с.
2. Керниган Б., Ритчи Д. Язык программирования Си. Москва: Финансы и статистика, 1992. – 272 с.
3. Подбельский В. В., Фомин С. С. Программирование на языке СИ. – Москва: Финансы и статистика, 1999. – 600 с.
4. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и системы. Условные обозначения и правила выполнения.
5. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.
6. Подбельский В. В. Язык С. Москва: Финансы и статистика, 2000. – 560 с.
7. Пол Ирэ. Объектно-ориентированное программирование с использованием С: Пер. с англ. - Киев: НИПФ “ДиаСофт Лтд, 1995.
8. Цимбал А.А. и др. Turbo C++, язык и его применение. - М.: Джен Ай Лтд, 1993, - 512с.
9. Bjarne Stroustrup The C++ Programming language, Addison Weasley, 1986.
10. Эллис М., Строуструп Б. Справочное руководство по языку С++ с комментариями: Пер. с англ. - Москва: Мир, 1992. 445с.
11. Стенли Б. Липпман. С++ для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992, 304-345сс.
12. Бруно Бабэ. Просто и ясно о Borland C++: Пер. с англ. - Москва: БИНОМ, 1994. 400с.
13. Фейсон Т. Объектно-ориентированное программирование на Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996. 544с.
14. Сван Т. Освоение Borland C++ 4.5: Пер. с англ. - Киев: Диалектика, 1996. 544с.
15. Шилдт Г. Самоучитель C++: Пер. с англ. - Санкт-Петербург: BHV-Санкт-Петербург, 1998. 620с.
16. Сэвитч У. С++ в примерах: Пер. с англ. - Москва: ЭКОМ, 1997. 736с.
17. Джамса К. Учимся программировать на языке С++: Пер. с англ. - Москва: Мир, 1997. 320с.
18. Скляров В.А. Язык С++ и объектно-ориентированное программирование: Справочное издание. - Минск: Вышэйшая школа, 1997. 480с.
19. Дейтел Х., Дейтел П. Как программировать на С++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с.

Додаток 1

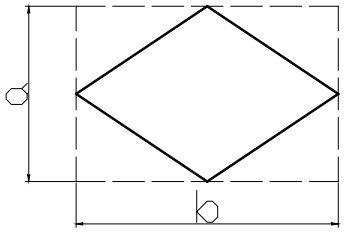
Виконання схем алгоритмів [4]

Основні символи, що використовуються при побудові блок-схем

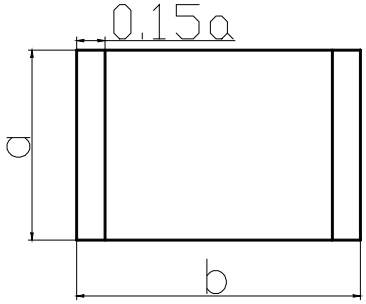
1. Процес – виконання операції чи групи операцій, у результаті якого змінюється значення, форма представлення чи розташування даних



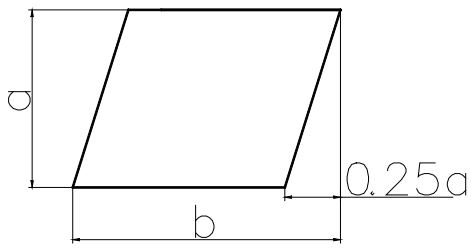
2. Рішення – вибір напрямку виконання алгоритму в залежності від деяких змінних умов



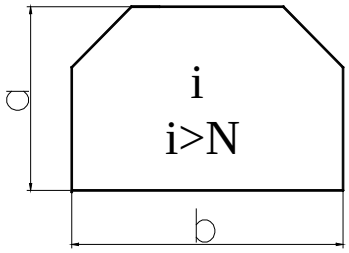
3. Визначений процес – використання окремо описаних алгоритмів чи програм (бібліотечні функції)



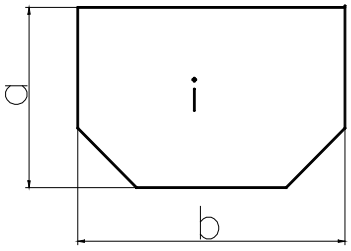
4. Символ вводу-виводу даних:



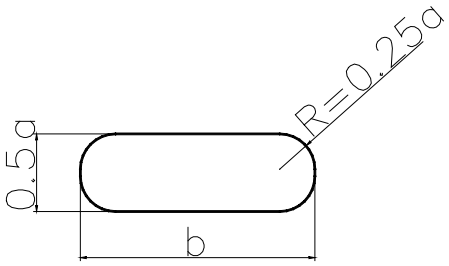
5. Символ початку циклу:



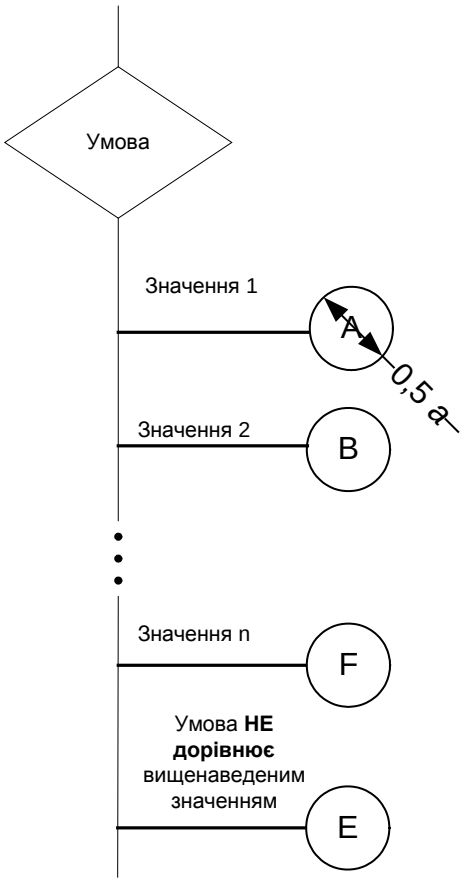
6. Символ кінця циклу:



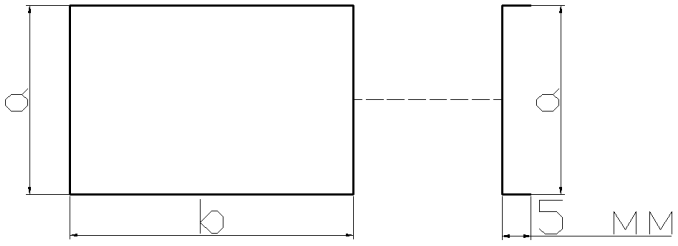
7. Символ початку, кінця, призупинення програми:



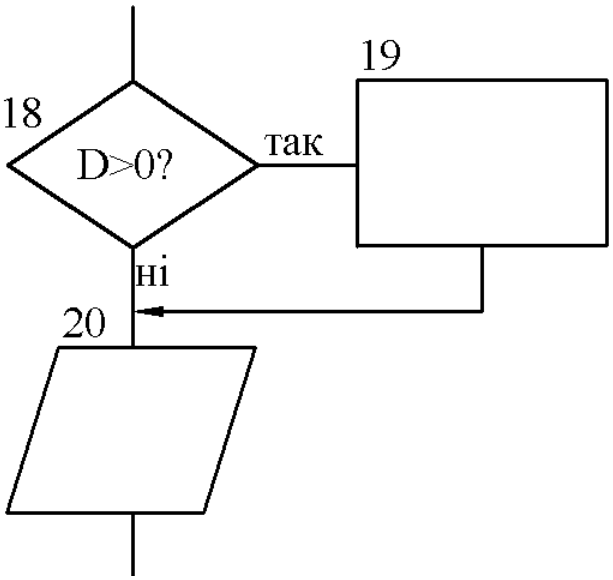
8. Якщо розгалужень більше двох, то використовується наступна схема:



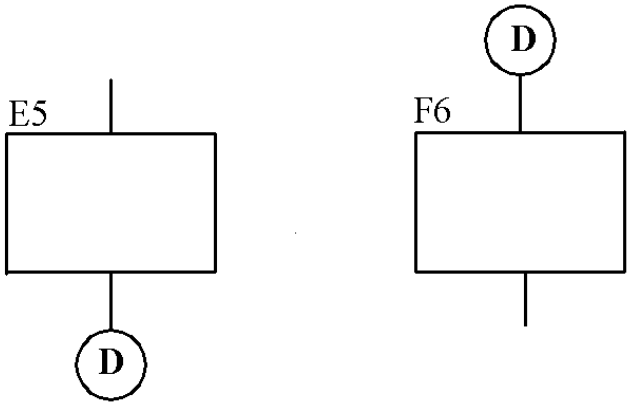
9. Коментарі до схеми:



10. Фрагмент схеми з розгалуженням

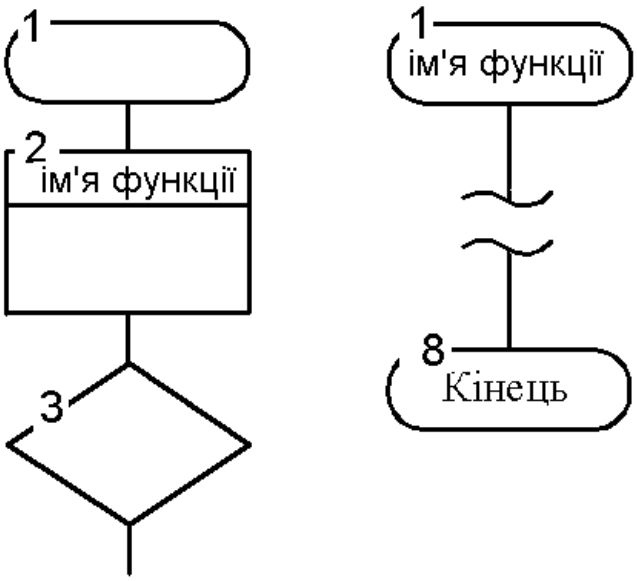


11. Використання з′єднувача



12. Визначений процес – використання раніше створених розробником функцій

13. Схеми на кількох рівнях



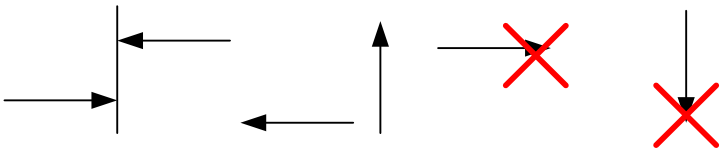
При виконанні схем алгоритмів основний розмір a повинний бути обраний з ряду 10, 15, 20 мм. Допускається збільшити це значення на число, кратне 5. Розмір *b* складає 1,5*a*, при ручному виконанні схем допускається *b* = 2*a*.

Символи нумеруються порядковими номерами чи координатами зон (арабські цифри або великі літери латинського алфавіту).

При ручному виконанні схеми в її межах допускається застосовувати не більш двох суміжних типорозмірів символів.

Лінії потоків повинні бути паралельні лініям зовнішньої рамки схеми.

Напрямок лінії потоку згори донизу і зліва направо стрілкою НЕ позначати. В інших випадках стрілки обов’язкові. (10)



Відстань між окремими символами схеми повинна бути не менше 10 мм.

Для пояснення процесів всередині символів розміщують короткі записи. Для зручності читання схем використовують коментарі до символів чи ліній потоку.

У межах одного аркуша лінії потоку можна розривати із застосуванням з’єднувачів, усередині яких міститься їх ідентифікатор. (11)

Можливі варіанти відображення ходу рішення при кількості можливих виходів два і більше показані на схемах 8, 9.

Якщо ж лініями потоку з’єднують символи на різних аркушах, то до кожного з′єднувача додається коментар – з якої сторінки, або до якої сторінки.

Схеми можуть бути побудовані на кількох рівнях.

Функції, які деталізуються, на крупних схемах позначаються прямокутником (12), в середині якого проведена горизонтальна лінія на 1/3 висоти символу. Посередині над цією лінією пишеться назва деталізованої функції.

В символі початку деталізованої функції пишеться її назва.

Приклади оформлення схем на більш поширенні комбінації при описі алгоритмів програм:

1. Виконання по умові

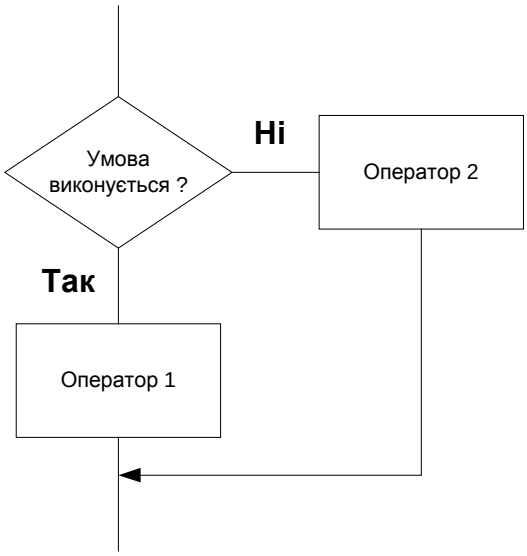
Умовний оператор *if*

if (вираз) оператор 1;

[else оператор2;]

Якщо ***вираз*** $\neq 0 \Rightarrow$ виконується ***оператор 1***

Якщо ***вираз*** $= 0 \Rightarrow$ виконується ***оператор 2***



switch *(вираз цілого типу)*

{

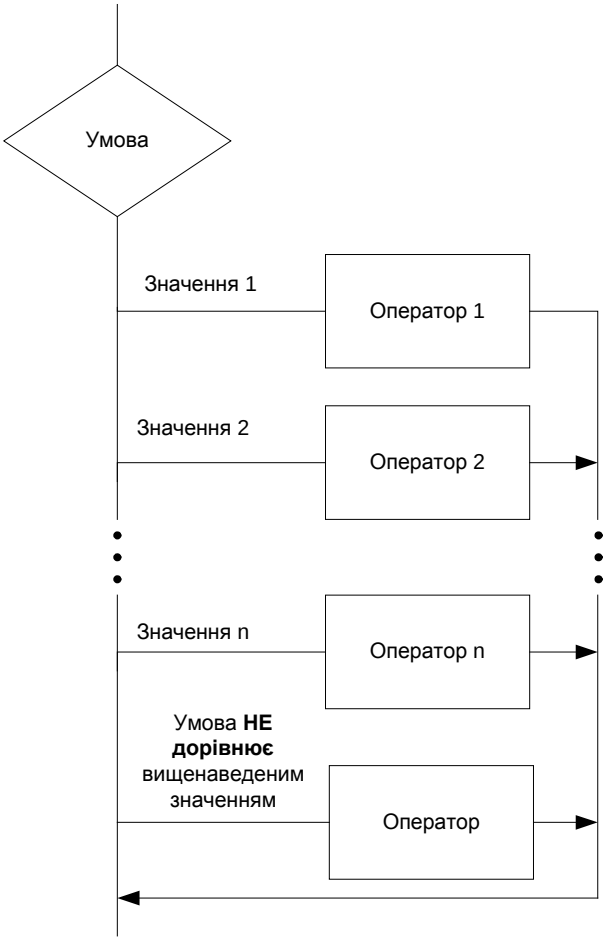
case *значення 1****: [****оператор 1****; break;] // вираз = значення 1***

case *значення 2****: [****оператор 2****; break;] // вираз = значення 2***

case *значення* n***: [****оператор n****; break;] // вираз = значення n***

[default :*оператор n+1****;] // вираз ≠ НІ одному з вищенаведених значень***

}



2. Циклічне виконання

Цикл із передумовою while

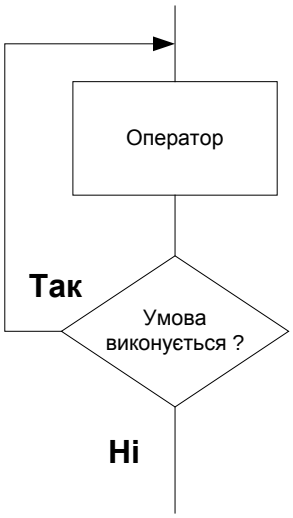
Оператор має структуру:

while (умова) оператор;

Цикл з післяумовою do-while

Коли ж оператори циклу повинні виконуватися принаймні один раз, то можна використати варіант:

do { оператор} while (умова)



Оператор циклу for

Має вигляд:

for (*вираз 1****;*** *вираз 2****;*** *вираз 3****) оператор;***

Виконується в такий спосіб:

вираз 1;

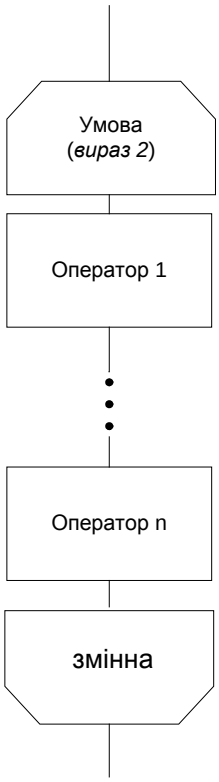
while (*вираз 2****)***

{*оператор; вираз 3;****}***

Вираз 1 виконується лише ***один*** раз.

Вираз 2 еквівалентно ***умові*** закінчення циклу.

Вираз 3 виконується ***щораз*** у циклі.



Додаток 2

Приклад виконання звіту

КПІ ім. Ігоря Сікорського

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіт до комп‘ютерного практикуму з курсу

“Програмування 1”

Прийняв
доцент кафедри ТК
Лісовиченко О.І.
“...” ..... 20xx р.

Виконав
Студент групи ІК-ХХ
Іванов Х.Х.

Київ 20xx

Комп‘ютерний практикум №1

Тема: *Програмування розгалужених алгоритмів*

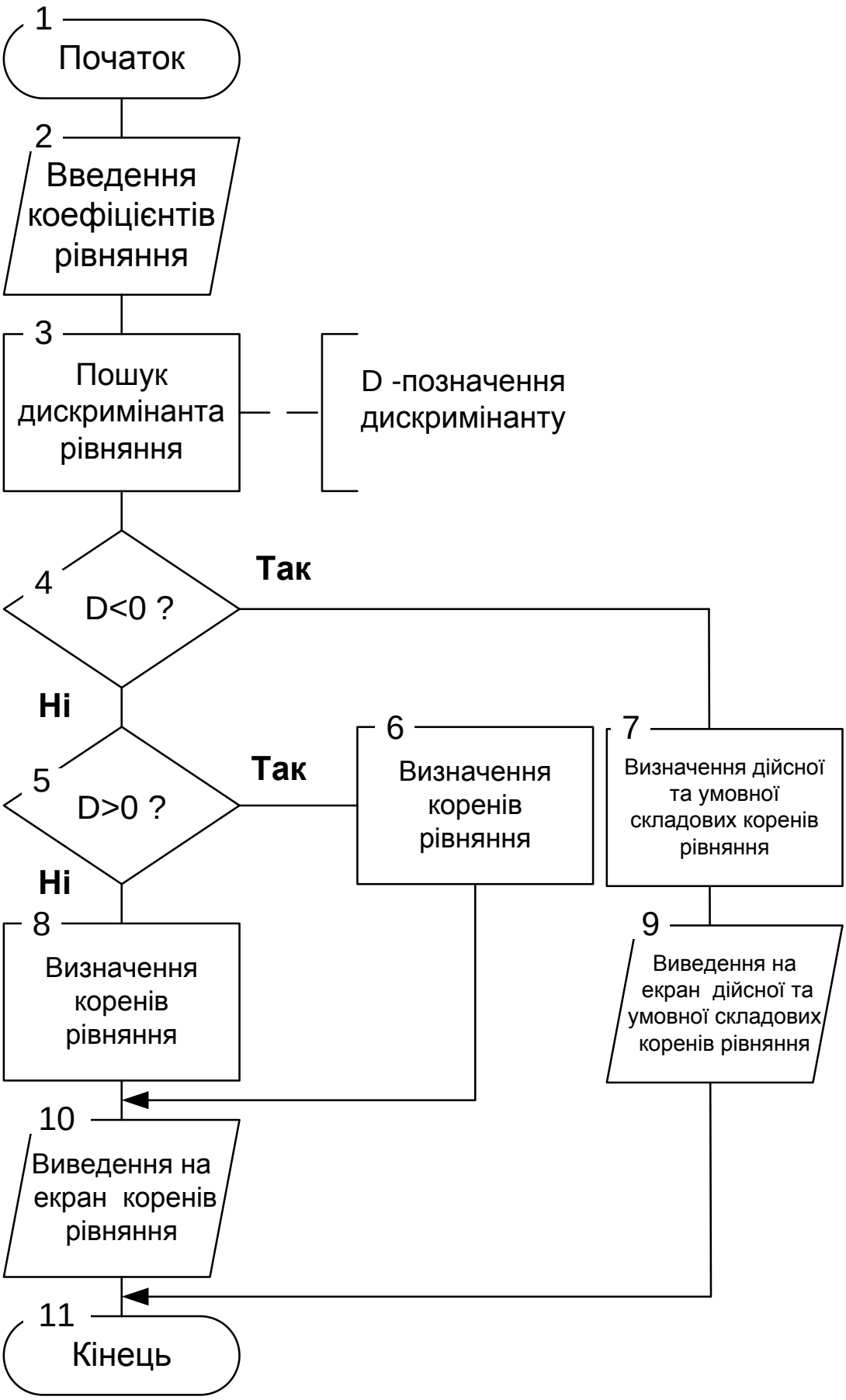
Завдання:

Скласти програму розв′язання квадратного рівняння.

Текст програми

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main()
{
  float x1,x2,re,im,a,b,c,d,f;
  clrscr();
  printf("Введіть коефіцієнти квадратного рівняння\n");
  printf("a = ");
  scanf("%f",&a);
  printf("b = ");
  scanf("%f",&b);
  printf("c = ");
  scanf("%f",&c);
  d=b*b-4*a*c;
  if (d<0)
  {  f=sqrt(-d);
     re=-b/(2*a);
     im=f/(2*a);
     printf("re = %f\nim = %f",re,im);
   }
   else
   { if (d>0) f=sqrt(d);
     else if (d=0) f=0;
     x1=(-b+f)/(2*a);
     x2=(-b-f)/(2*a);
     printf("x1 = %f\nx2 = %f",x1,x2);
    }
  getch();
  return 0;
}
```

Схема до програми



Введені та одержані результати:

```
Введіть коефіцієнти квадратного рівняння
a = 1
b = 1
c = 1
re = -0.500000
im = 0.866025
```

```
Введіть коефіцієнти квадратного рівняння
a = 3
b = 2
c = -1
x1 = 0.333333
x2 = -1.000000
```

```
Введіть коефіцієнти квадратного рівняння
a = 5
b = 4
c = -2
x1 = 0.348332
x2 = -1.148332
```

```
Введіть коефіцієнти квадратного рівняння
a = 3
b = 2
c = 4
re = -0.333333
im = 1.105542
```

Теоретичні розрахунки:

$a := 1 \quad b := 1 \quad c := 1 \quad D := b \cdot b - 4 \cdot a \cdot c \quad D = -3$

$x1 := \frac{-b + \sqrt{D}}{2 \cdot a} \qquad x2 := \frac{-b - \sqrt{D}}{2 \cdot a}$

$x1 = -0.5 + 0.866i \qquad x2 = -0.5 - 0.866i$

$a := 3 \quad b := 2 \quad c := -1 \quad D := b \cdot b - 4 \cdot a \cdot c \quad D = 16$

$x1 := \frac{-b + \sqrt{D}}{2 \cdot a} \qquad x2 := \frac{-b - \sqrt{D}}{2 \cdot a}$

$x1 = 0.333 \qquad x2 = -1$

$a := 5 \quad b := 4 \quad c := -2 \quad D := b \cdot b - 4 \cdot a \cdot c \quad D = 56$

$x1 := \frac{-b + \sqrt{D}}{2 \cdot a} \qquad x2 := \frac{-b - \sqrt{D}}{2 \cdot a}$

$x1 = 0.348 \qquad x2 = -1.148$

$a := 3 \quad b := 2 \quad c := 4 \quad D := b \cdot b - 4 \cdot a \cdot c \quad D = -44$

$x1 := \frac{-b + \sqrt{D}}{2 \cdot a} \qquad x2 := \frac{-b - \sqrt{D}}{2 \cdot a}$

$x1 = -0.333 + 1.106i \quad x2 = -0.333 - 1.106i$

Висновки: Теоретичні розрахунки відповідають отриманим. Програма працює корректно. Програма вирішує поставлене завдання.