Практична робота № 1
Алгоритмічний опис лінійної структури і програмування на мові С найпростіших задач

Мета роботи.

Дати поняття про зображення алгоритмів за допомогою блок-схем, сформувати у студентів навики складання найпростіших блок-схем алгоритмів лінійної структури. Познайомити студентів з написання найпростіших програм на мові С і використання стандартних функцій *printf* та *scanf*. Сформувати у студентів навики використання шаблонів форматного рядка.

Методичні вказівки

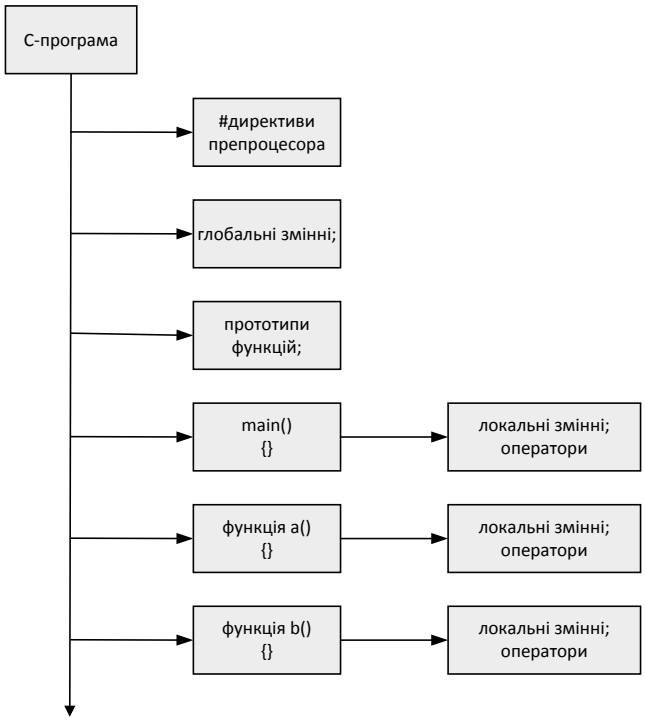
Коротко нагадати студентам про правила виконання блок-схем згідно із ДСТУ ISO 5807:2016 Обробляння інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів (ISO 5807:1985, IDT)., а саме:

- основні графічні символи (процес, рішення, ввід-вивід, початок-кінець, з'єднувачі);
- розміри графічних символів;
- правила з'єднання графічних символів;

і нагадати, що блок-схема алгоритму залежить від виконавця алгоритму.

Також нагадати про три структури, з яких будуються блок-схеми алгоритмів: лінійна структура, структура з розгалуженням і циклічна структура.

Структура програми, написаної мовою програмування С, має вигляд:



Консольний ввід-вивід з клавіатури.

При запуску програми на С автоматично відкриваються п'ять потоків, основними з яких є наступні:

- Стандартний потік введення `stdin`;
- Стандартний потік виводу `stdout`;
- Стандартний потік виводу повідомлень про помилки `stderr`.

Стандартний потік введення `stdin` за замовчуванням відповідає клавіатурі, а потоки `stdout` і `stderr` - екрану монітора.

Для форматованого виводу інформації в консольному режимі є функція `printf()`. Загальний вигляд функції є такий:

```
int printf( const char* format[, argument]... );
```

Тут `format` задає пояснювальний текст та вигляд значень змінних, імена яких задає параметр `argument` (список змінних). Параметр список змінних не є обов'язковим і являє собою послідовність розділених комами змінних, значення яких виводяться.

Специфікатор формату задає вигляд виведеного результату.

Специфікатор формату для `printf()`:

```
%[прапорці] [ширина] [.точність] [{ h | l | L }]тип
```

У загальному вигляді функція `scanf()`виглядає так:

```
int scanf( const char* format[, argument]... );
scanf( формат, &змінна );
```

де: `format` – рядок специфікаторів формату у подвійних лапках, `argument` – це ім'я змінної, значення якої вводиться. Перед іменем змінної треба ставити знак `&`, що означає операцію отримання адреси змінної у пам'яті.

Специфікатор формату для `scanf()`:

```
%[*] [ширина] [{h | l | L}]тип
```

Контрольні запитання

1) Яка структура програми на мові С?
2) Які специфікатори формату функції **`printf`**?
3) Які специфікатори формату функції **`scanf`**?
4) Як називається основна функція мови С?
5) З яких основних графічних символів складаються блок-схеми?
6) Яке призначення символу *процес*?
7) Яке призначення символу *рішення*?
8) Яке призначення символу *ввід-вивід*?
9) Яке призначення символу *початок-кінець* (*вхід-вихід*)?
10) Які правила проставляння стрілок на *лініях потоку*?
11) Які структури алгоритмів вважаються *лінійними структурами*?
12) Яке призначення графічного символу *зумовлений процес*?
13) Який графічний символ використовується для позначення *з'єднувача*?
14) Які правила запису коментарів на блок-схемах?

Задачі для домашнього завдання

1) Ввести з клавіатури за допомогою функції **`scanf`** наступні значення:
   5.5, 555, -555555, символ '5'.
2) Вивести на екран з різною шириною виводу за допомогою функції **`printf`** наступні значення:
   1.1, 111, -111111, символ '1'.

Практична робота № 2
Алгоритмічний опис структур з розгалуженням і програмування задач, в яких використовуються умовні оператори.

Мета роботи.

Закріпити знання про зображення алгоритмів з допомогою блок-схем, отримані на попередньому занятті. Сформувати у студентів навики складання блок-схем алгоритмів структури з розгалуженням.

Методичні вказівки

Почати доцільно зі складання блок схеми розв'язку задачі знаходження найменшого з двох чисел *a* і *b*.

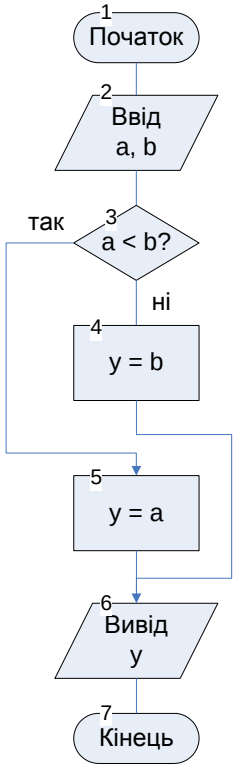


Рис 2. y=min(a,b)

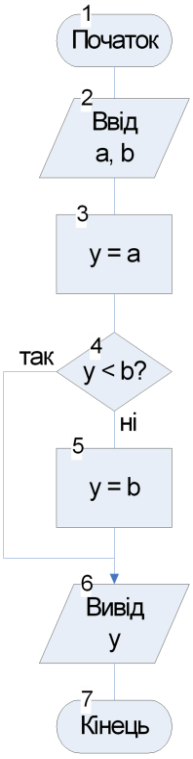


Рис 3. y=min(a,b)

Блок-схема алгоритму може мати такий вигляд:

Слід звернути увагу студентів на:
a) розміщення символів;
b) правильне (згідно з ДСТУ) позначення стрілок на блок-схемі.;
c) ще раз нагадати про розміщення надписів «так», «ні» біля виходів символу «рішення».

Запропонувати студентам самостійно скласти блок-схему для знаходження найбільшого значення двох чисел *a* і *b*. Проаналізувати виконані блок-схеми, вказати на помилки, які, в основному будуть стосуватися пункту *a*.

Розроблена блок-схема рис.2 не є оптимальною. В більшості підручників (переважно шкільних) власне так її і зображають.

Блок-схема рис. 2 правильно розв'язує поставлену задачу, але не є найкращим рішенням. Оптимальнішим буде розв'язок представлений на рис. 3. Підкреслити, що спосіб яким побудована блок-схема рис. 3 у порівнянні з блок-схемою рис. 2 є певним шаблоном, які використовує досвідчений програміст, щоб не «придумувати», яким чином складати нову блок-схему.

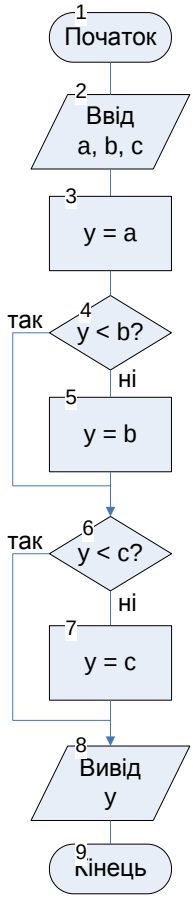


Рис 4. y=min(a,b,c)

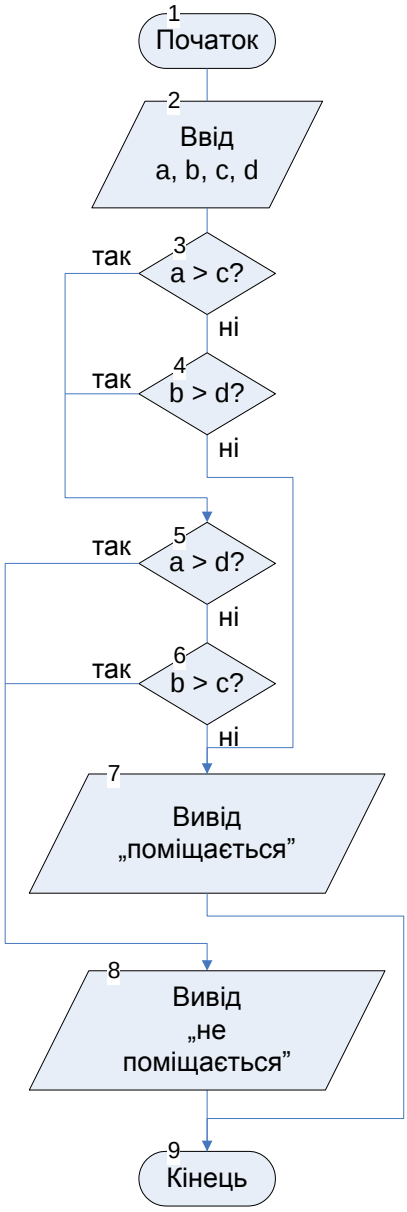


Рис 5.Розміщення прямокутників

Також розказати, що коли потрібно буде знайти найменший (найбільший) елемент в масиві значень (з чим студенти будуть знайомитися в наступних заняттях), таку блок-схему можна будувати цим же способом.

Наступною задачею має бути блок-схема для знаходження найменшого із трьох чисел *a*, *b* і *c*. Спочатку запропонувати зробити цю задачу самостійно і проаналізувати виконані студентами блок-схеми, а після цього запропонувати оптимальний варіант (рис. 4).

Як задачу для розгляду питання побудови складніших блок-схем з розгалуженням можна взяти наступну.

Задані дійсні додатні числа *a, b, c, d.* Визначити, чи можна прямокутник зі сторонами *a, b* помістити всередині прямокутника зі сторонами *c, d* так, кожна із сторін одного прямокутника була паралельна чи перпендикулярна кожній стороні іншого.

Розв’зати цю задачу можна двома способами. Перший – пряме порівняння, другий – з використанням підпрограми знаходження мінімуму та максимуму.

Пояснюючи розв’язок цієї задачі першим способом слід наголосити, що виконавець програма в машинному коді у символі «рішення» може виконувати лише одне порівняння і виконання логічних операцій «І», «АБО», тощо (що вірно для виконавця – програма на мові високого рівня) тут не допустиме.

Розв’язок задачі представлений на рис. 5. Звернути увагу студентів на те, що необхідність розміщення виходу «ні» в графічному символі «рішення» внизу символу, приводить до того, що необхідно використовувати логіку «від протилежного». Якщо не виконується умова *a* більше *c*, і не виконується умова *b* більше *d*, то це еквівалентне до твердження: виконується умова *a* менше-рівне *c* і умова *b* менше-рівне *d*.

Розв’язок задачі другим способом (з використанням підпрограми) показано на рис 6, блок-схема підпрограми на рис. 7.

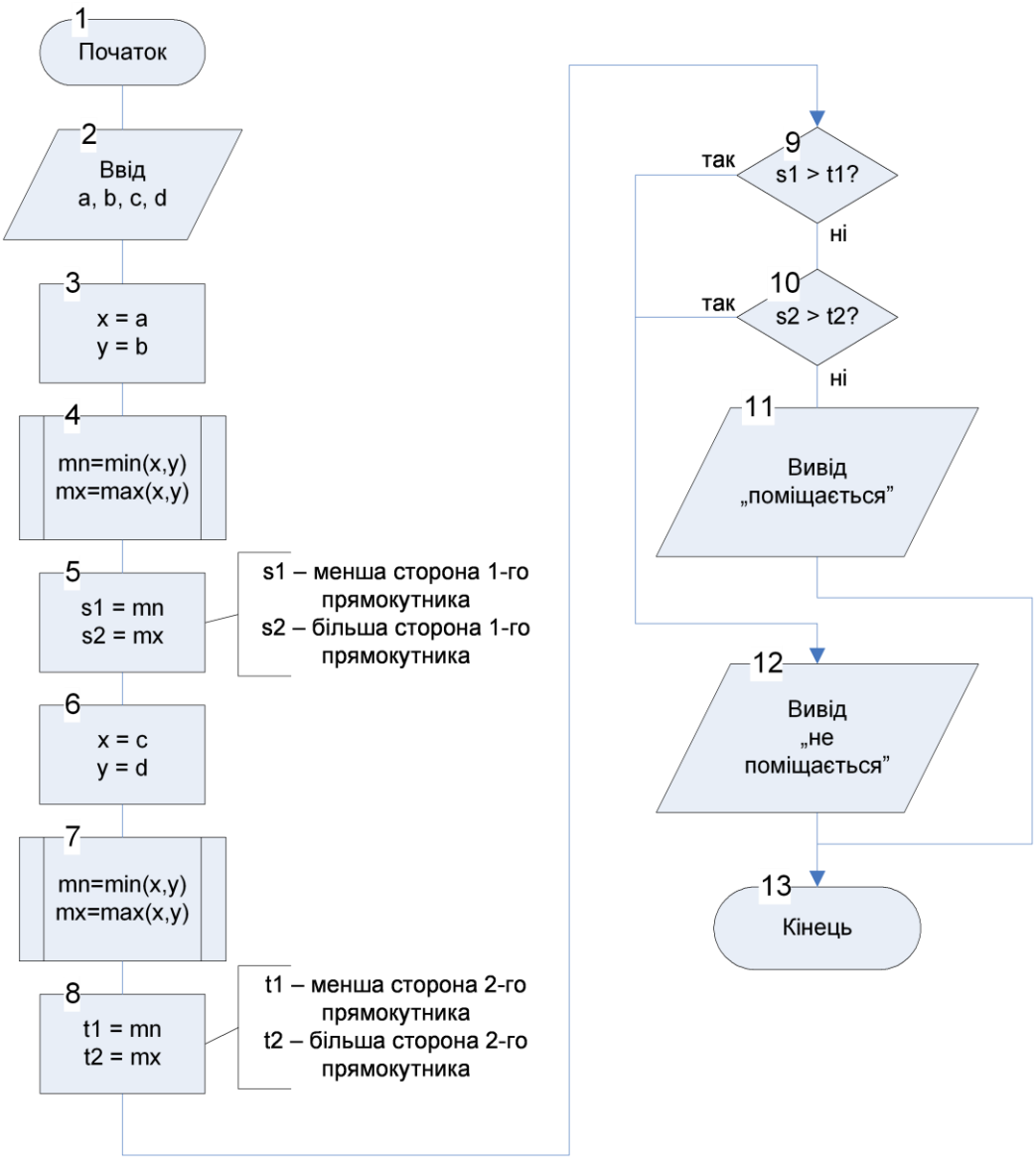


Рис 6. Розміщення прямокутників (2-й спосіб)

Перед розглядом цієї блок-схеми познайомити студентів з графічним символом «зумовлений процес», який використовується в блок-схемах для зображення процесу виклику підпрограми і нагадати про використання коментарів в блок-схемах. Нагадати також про те, що лінії потоку не повинні перетинатися і про використання з'єднувачів (внутрішньо сторінкових і міжсторінкових).

Коротко проінформувати студентів про переваги та недоліки використання підпрограм і пояснити блок-схему рис. 6 і підпрограму рис. 7.

Завдання: дано дійсні числа A, B, C. Скласти схему алгоритму обчислення $Z = \max (A^2 + B, \min (B + 1, C))$.

Розв'язок: у даному прикладі можливо три варіанти відповіді:

або $A^2 + B$, або $B + 1$, або C. І вибір буде виконаний лише за результатами перевірки умов.

Обчислення величини Z виконується в два етапи. На першому етапі вибираємо min з двох величин $B + 1$ і C. Результат вибору позначимо через проміжну змінну R, тобто $R = \min (B + 1, C)$.

Тоді формула для обчислення Z буде мати вигляд

$Z = \max(A^2 + B, R)$.

Алгоритм обчислення матиме вигляд:

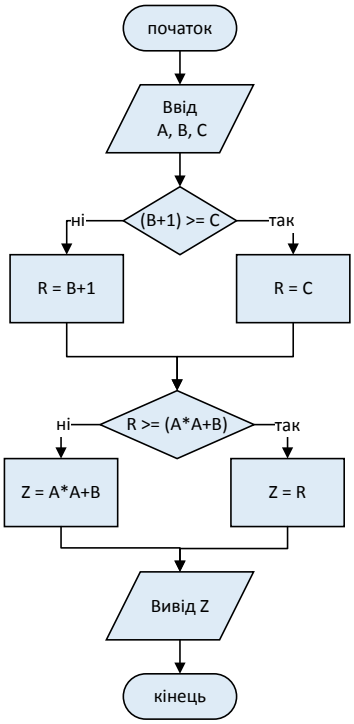


Рис. Блок-схема алгоритму обчислення $Z = max\ (A^2 + B, min\ (B + 1, C))$.

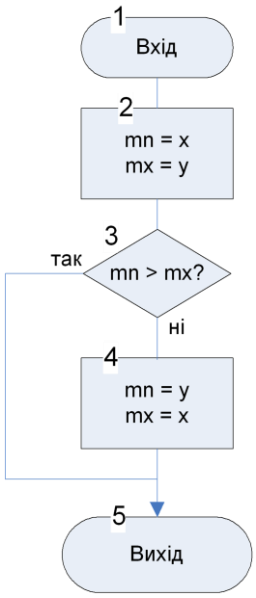


Рис. 7. Підпрограма

Контрольні запитання

1. З яких основних графічних символів складаються блок-схеми?
2. Яке призначення символа *процес*?
3. Яке призначення символа *рішення*?
4. Яке призначення символа *ввід-вивід*?
5. Яке призначення символа *початок-кінець* (*вхід-вихід*)?
6. Які правила проставляння стрілок на *лініях потоку*?
7. Які структури алгоритмів вважаються *лінійними структурами*?
8. Які структури алгоритмів вважаються *структурами з розгалуженням*?
9. Яке призначення графічного символа *зумовлений процес*?
10. Який графічний символ використовується для позначення *з'єднувача*?
11. Які правила запису коментарів на блок-схемах?
12. Який синтаксис запису оператора **`if`**?

Задачі для домашнього завдання.

1. Задане дійсне число *a*. Обчислити *f(a)*, якщо

а) $f(x) = \begin{cases} x^2 & \text{при} -2 \le x < 2, \\ 4 & \text{в інших випадках;} \end{cases}$

б) $f(x)=\begin{cases}0 & \text{при } x \le 0,\\ x & \text{при } 0 < x \le 1,\\ x^4 & \text{в решті випадків;}\end{cases}$

в) $f(x)=\begin{cases}0 & \text{при } x \le 0,\\ x^2 - x & \text{при } 0 < x \le 1,\\ x^2 - \sin \pi x^2 & \text{в решті випадків;}\end{cases}$

2. Задані дійсні числа *x, y*. Визначити, визначити які четверті Декартової площини координат належить точка із координатами (*x, y*).

3. Задані дійсні числа *a, b, c*. Перевірити, чи виконується нерівність $a<b<c$.

4. Задані три дійсних числа. Вибрати ті із них, які належать інтервалу (1, 3).

5. Задані дійсні додатні числа *a, b, c, x, y*. Визначити чи пройде цегла із ребрами *a, b, c* в прямокутний отвір зі сторонами *x, y*. Просовувати цеглу в отвір дозволяється тільки так, щоб кожне її ребро було паралельним чи перпендикулярним кожній із сторін отвору.

Задачу зробити двома способами: прямим порівнянням, повертаючи цеглу різними гранями і з використання підпрограм, які впорядковують по зростанню ребра цегли і сторони отвору.

Практична робота № 3
Алгоритмічний опис циклічних структур і програмування задач, в яких використовуються оператори циклу.

Мета роботи.

Закріпити знання про зображення алгоритмів з допомогою блок-схем, отримані на попередніх заняттях. Сформувати у студентів навики складання блок-схем алгоритмів циклічної структури.

Методичні вказівки.

Спочатку згадати про дві структури циклічних алгоритмів – циклічну *структуру з передумовою* і *циклічну структуру з післяумовою*. Так як студенти часто плутаються між цими двома структурами, щоб краще виробити навики складання блок-схем циклічних структур на одній з них. На практичних заняттях вибрати як основну циклічну структуру з післяумовою і лише після твердого засвоєння цієї структури можна почати використовувати і іншу.

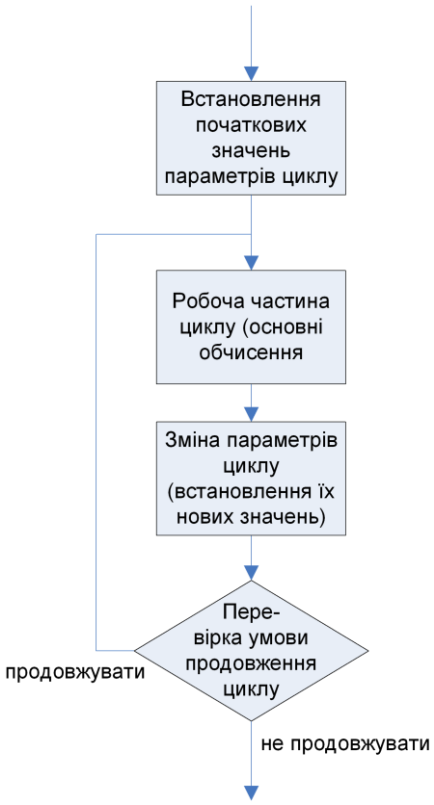


Рис.8. Шаблон циклу з післяумовою

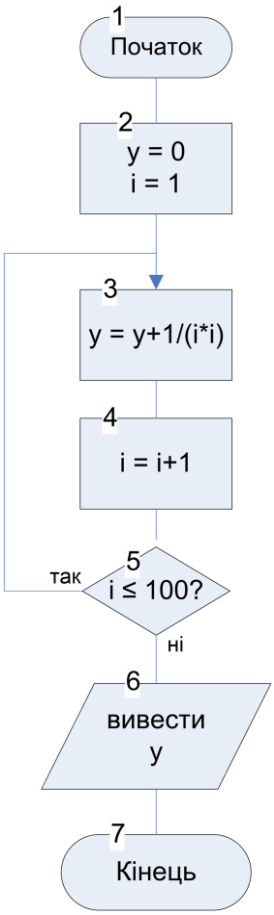


Рис. 9. Сума 1

Дати студентам типову схему (шаблон) побудови блок-схеми циклічної структури з післяумовою (рис. 8). Він складається з блоку підготовки параметрів циклу (блоку ініціалізації), блоку робочої частини циклу, блоку зміни параметрів циклу (блок модифікації) та блоку керування циклом.

Коротко згадати чим відрізняється шаблон циклу з передумовою від циклу з післяумовою.

Студентів потрібно привчити до думки, що при складанні блок-схеми циклічної структури слід спочатку намалювати порожні графічні символи шаблону, а після цього їх заповнювати За рідкими винятками може виявитися порожнім блок модифікації. Це буває коли зміна параметра (параметрів) циклу здійснюється в робочій частині. Але студентів потрібно спонукати явно виділяти операції модифікації параметрів.

Першою задачею доцільно взяти таку, на якій найочевидніше буде продемонструвати використання шаблону рис. 8. Наприклад, обчислити

$$\sum_{i=1}^{100} \frac{1}{i^2} \qquad (1)$$

Блок-схема рішення задачі представлена на рис. 9.

Перед складанням блок-схеми спочатку протрібно провести аналіз задачі. Формулу (1) «розписуємо».

$$y = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{100^2} \qquad (2)$$

З формули (2) витікає, що обчислення *y* полягає в послідовному додаванні доданків $1/1^2$, $1/2^2$, $1/3^2$ і т. д. до суми. Тобто, якщо в блоці ініціалізаціїї встановити початкове значення суми рівне нулю, а в блоці «робоча частина циклу» додавати до нагромадженої на попередніх кроках циклу суми черговий доданок, то ми реалізуємо обчислення за формулою (2). В блоці модифікації кожен раз необхідно збільшувати значення параметру циклу (змінної *i*) на одиницю. А в блоці керування циклом слід перевіряти чи змінна *i* не перевищила значення 100. Знову нагадати студентам, що тут використовуємо логіку «від протилежного».

Також необхідно звернути увагу студентів на те, що в графічних символах 3 і 4 (як і решту) символ «=» позначає не знак рівності, а операцію присвоєння:

i = i +1

означає, що з пам'яті комп'ютера вибирається значення змінної *i* (старе значення *i*), збільшує на одиницю і записується у ту саму комірку пам'яті (нове значенню *i*).

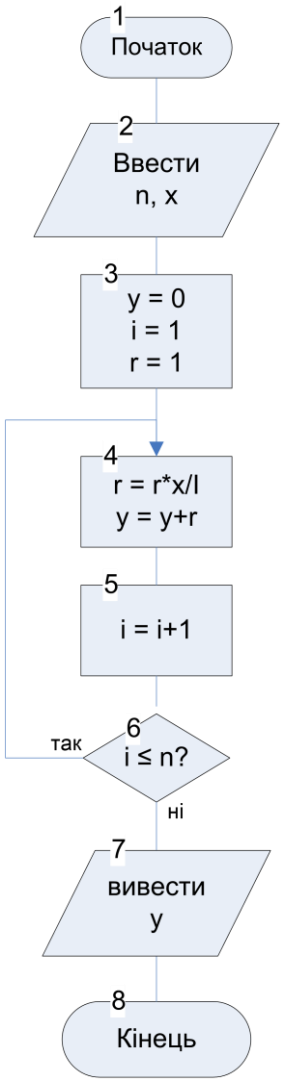


Рис.10. Сума 2

Другою задачею мала би вводитити дещо складніші елементи, які підкреслювали би особливості аналізу задачі, та особливості обчислень на комп'ютері. Нариклад, для заданого натурального числа *n* та дійсне числа *x*, обчислити:

$$\sum_{i=1}^{n} \frac{x^i}{i!} \qquad (3)$$

Як і в попередній задачі необхідно обчислити суму доданків. Але виконавець програма в машинному коді не може реалізувати ні операції піднесення до степеня, ні обчислення факторіалу. Тому операцію піднесення до степеня має бути реалізована через множення в циклі попереднього значення степеня на *x*, а фактріал *i!* – шляхом домноження попереднього значення факторіалу на наступне значення *i*.

Іншою особливістю цієї задачі є та, що при великих значеннях *n* і великих значеннях *x* пряме обчислення чисельника і знаменника може давати дуже великі значення і навіть переповнення розрядної сітки. Тому обчислення чергового значення доданку шляхом домноження попереднього його значення на *x* та ділення на *i* є кращим розв'язком цієї проблеми.

Блок-схема розв'язку другої задачі (формула (3)) представлена на рис. 10.

Запропонувати студентам скласти блок-схему розв'язку наступних задач:

1. Задане натуральне число *n*, обчислити добуток перших *n* співмножників.

а) $\frac{1}{2} \cdot \frac{3}{4} \cdot \frac{5}{6} \cdot \ldots$;

б) $\frac{1}{1} \cdot \frac{3}{2} \cdot \frac{5}{3} \cdot \ldots$ ..

1. Задане натуральне число *n*, обчислити:

$$\sqrt{2 + \sqrt{2 + \cdots + \sqrt{2}}} \qquad (n \text{ коренів})$$

Аналіз правильності виконання складеної блок-схеми алгоритму проводиться шляхом моделювання («прокручування») виконання блок-схеми. Блок-схема уявно покроково виконується із врахуванням розгалужень. Результат виконання порівнюється з поставленою задачею. Пояснити студентам на декількох блок-схемах.

Контрольні запитання

1) Які структура алгоритмів називають циклічними?
2) За яким шаблоном можна будувати циклічну структуру алгоритмів?
3) Яке призначення блоку ініціалізації шаблону циклічної структури алгоритму?
4) Яке призначення блоку «робоча частина» шаблону циклічної структури алгоритму?
5) Яке призначення блоку модифікації шаблону циклічної структури алгоритму?
6) Яке призначення блоку управління циклом шаблону циклічної структури алгоритму?
7) Який синтаксис запису оператора **`for`**?
8) Який синтаксис запису оператора **`while`**?
9) Який синтаксис запису оператора **`do-while`**?

Задачі для домашнього завдання

1. Задане натуральне число *n*, обчислити:

$$\sqrt{3+\sqrt{6+\cdots+\sqrt{3(n-1)+\sqrt{3n}}}}$$

2. Задане натуральне *n*, дійсне *x*. Обчислити:

а) $\sin x+\sin^2 x+\ldots+\sin^n x$;

б) $\sin x+\sin x^2+\ldots+\sin x^n$;

в) $\sin x+\sin\sin x+\ldots+\sin\sin\ldots\sin x$ (в останньому доданку *n* синусів).

3. Задане натуральне *n*, дійсне *a*. Обчислити:

$$\frac{1}{a}+\frac{1}{a^2}+\frac{1}{a^4}+\ldots+\frac{1}{a^{2^n}} \ .$$

Практична робота № 4
Алгоритмічний опис вкладених циклічних структур. Програмування на С задач, в яких використовуються власні функції..

Мета роботи.

Познайомити студентів з вкладеними циклічними структурами, функціями, локальними і глобальними змінними, а також специфікаторами класів пам'яті у мові С. Сформувати у студентів навики написання програм на мові С з використанням власних функцій.

Методичні вказівки.

Так як формування навиків складання блок-схем алгоритмів циклічної структури є надзвичайно важливим, закріпленню цих навиків необхідно приділити достатньо часу. Можна розглянути, наприклад, таку задачу:

Обчислити:

$$y = \cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{5+\cfrac{1}{\ddots}}}}$$

$$101+\cfrac{1}{105}$$

Аналіз цієї задачі проводимо таким чином: щоб розділити чисельник на знаменник, потрібно мати його значення. Тобто знаменник потрібно обчислити раніше. Тому обчислення ланцюгового дробу потрібно почати з «найнижчого» дробу – 1/105.

Розв'язок задачі представлений на рис. 11. Слід підкреслити студентам, що незважаючи на різні задачі, всі блок-схеми будуються за тим самим шаблоном (див. блок-схеми рис. 9, рис. 10, рис. 11).

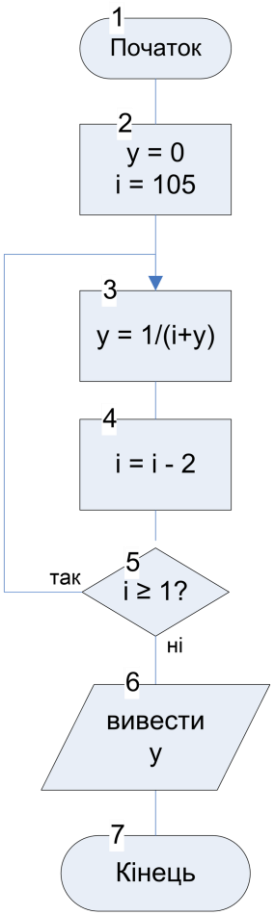


Рис.11. Ланцюговий дріб

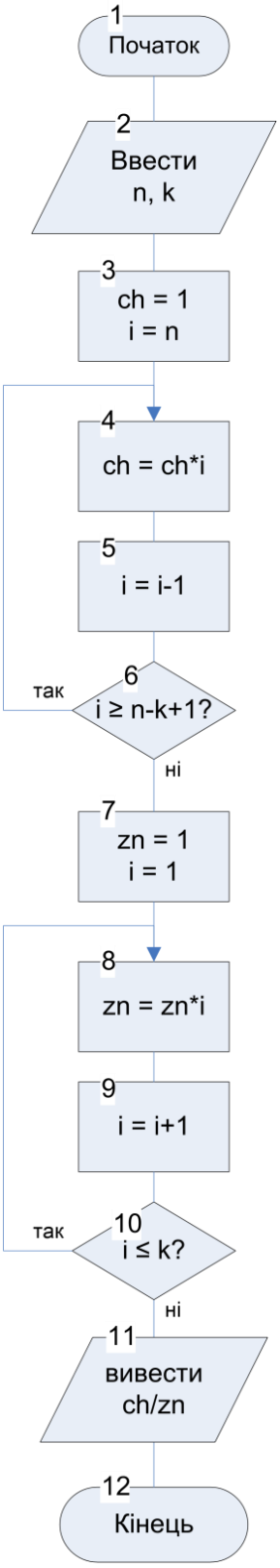


Рис.12. Дріб

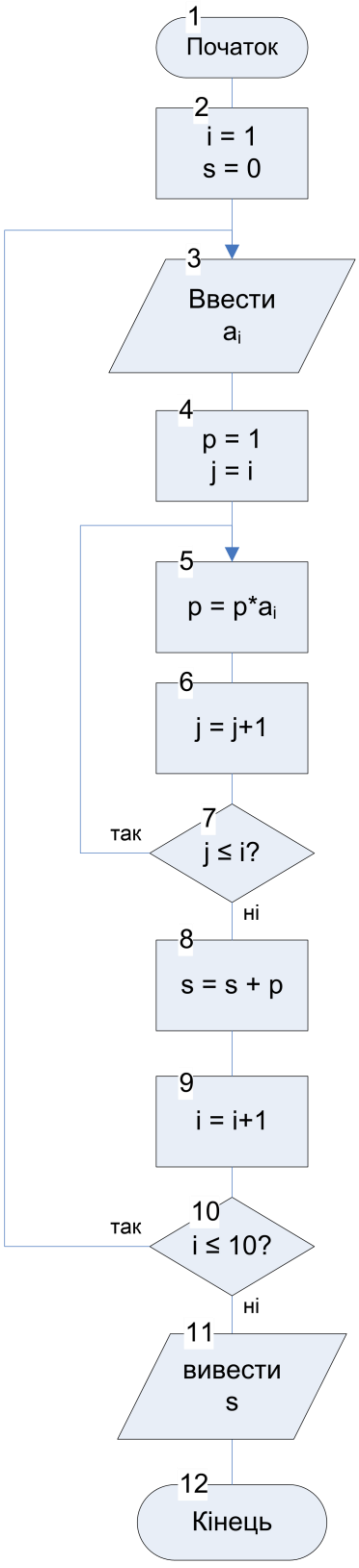


Рис. 13. Вкладений цикл

Ще однією задачею може бути наступна:

Задані цілі числа n, k ($n \geq k \geq 0$), обчислити:

$$\frac{n(n-1)(n-2)\ldots(n-k+1)}{k!}$$

Аналіз цієї задачі показує, що при обчисленні чисельника і знаменника множиться різна кількість співмножників, а, значить чисельник і знаменник мають обчислюватися в різних циклах.

Вкладені цикли отримуються, коли робочою частиною циклічної структури є інша циклічна структура.

Як приклад можна взяти таку задачу:

Задані дійсні числа a_1, a_2, . . . , a_{10}. Обчислити

$$a_1 + a_2^2 + \ldots + a_{10}^{10}$$

Тут явно прослідковується два цикли: у зовнішньому виконується підрахунок суми доданків, у внутрішньому – обчислення степені.

Блок-схема розв'язку задачі представлена на рис. 13. Робочою частиною зовнішнього циклу є блоки 3 – 8, які включають внутрішній цикл (блоки 4 – 7) та блок вводу (3) і блок 8.

Варто детально розглянути блок-схему задачі з домашнього завдання про досконалі числа.

Блок-схема приведена на рис. 14. Підрахунок суми дільників числа здійснюється у внутрішньому циклі (блоки 4 – 8). В блоці 5 проводиться перевірка: якщо остача від ділення числа *i* на число *j* рівна нулю, то число *j* є дільником числа *i*. Найбільший дільник не може перевищувати половини числа *i*, тому кількість повторень внутрішнього циклу має бути рівна [*i/2*].

Робоча частина зовнішнього циклу включає внутрішній цикл та блоки 9 – 10, в яких перевіряється, чи дане число *i* є досконалим. В блоці ініціалізації встановлюємо *i* рівним 2 (а не 1), тому що за визначенням досконаєм є число, яке рівне сумі всіх своїх дільників, за винятком самого себе.

Між іншим, досконалих чисел є небагато: в діапазоні від 2 до 10000 їх лише чотири (6, 28, 496, 8128).

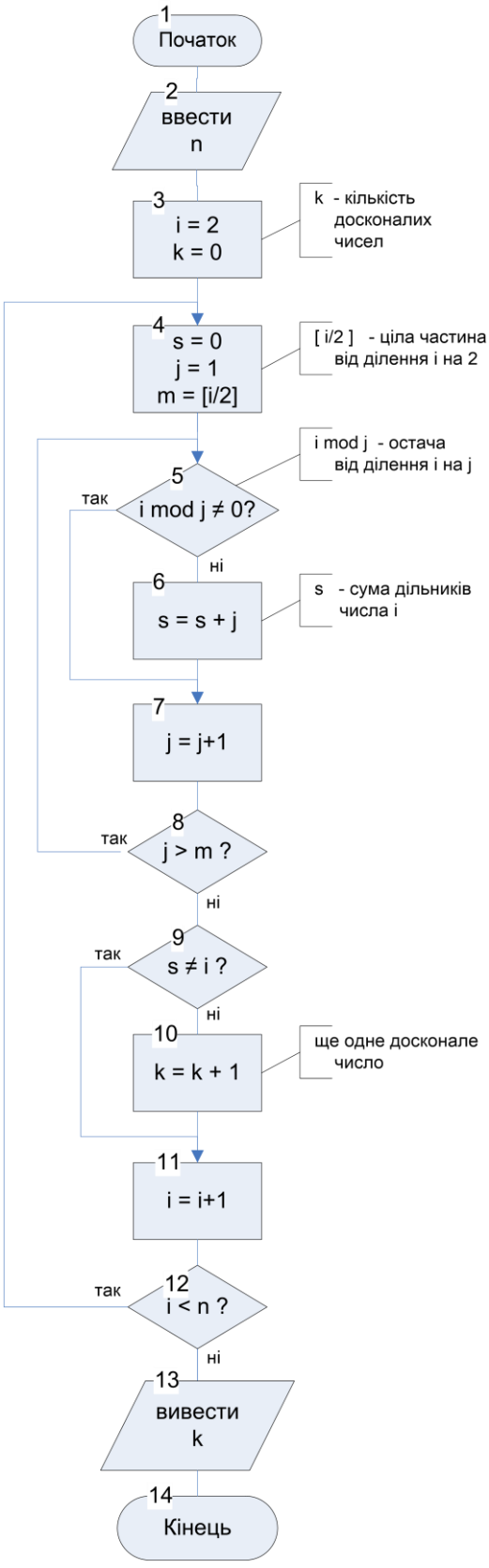


Рис.14. Досконалі числа

Функція – це незалежна іменована частина програми, яка може багаторазово викликатися з інших частин програми, маніпулювати даними та повертати результати. Кожна функція має власне ім'я, за яким здійснюють її виклик.

Створювана у програмі функція повинна бути *оголошеною* і *визначеною*. Оголошення функції має бути написаним у програмі раніш за її використання. Визначення може перебувати у будь-якому місці програми, за винятком тіла (середини) інших функцій. *Оголошення* функції складається з *прототипу* (чи *заголовка*) і має форму

```
[клас] <тип результату> <ім'я функції>
(<перелік формальних аргументів із зазначенням типу>);
```

Наприклад оголошення функції обчислення середньоарифметичного двох цілих чисел може мати видгляд

```
float seredne (int a, int b);
```

де `seredne` – ім'я функції;

`a` і `b` – формальні вхідні аргументи (параметри), які за виклику набувають значень фактичних параметрів;

`float` – тип функції, який безпосередньо є типом результату виконання операторів функції. Результат повертається оператором `return`.

Функція не може повертати, як результат, функцію чи масив, проте вона може повертати вказівник на функцію чи масив.

Окрім оголошення, кожна функція повинна мати визначення (реалізацію). *Визначення* функції, окрім заголовку, містить тіло функції (команди, які виконує функція):

<тип функції> <ім'я функції> (<перелік аргументів>)
{ <тіло функції> }

Наприклад, визначення функції `seredne` може бути таким:

```
float seredne (int a, int b)
{ float sr;
sr=(a+b)/2.0;
return sr;
}
```

Наведена функція обчислює `sr` – середньоарифметичне двох цілих чисел `a` та `b` і повертає його значення у точку виклику.

Якщо функції виконують певні обчислення й дії, які не потребують повертання результатів, за їхній тип вказують тип `void` (тобто порожній, без типу). У таких функціях оператор `return` може бути відсутнім чи записуватись без значення, яке повертається.

Інструкція для виклику функції складається з імені функції і ( ). Тут круглі дужки означають операцію виклику функції. Виклик функції *`seredne`* можна здійснити таким чином

```
        seredne ( x, y )
   чи
        seredne ( 20, 100 )
```

Контрольні запитання

1. Що таке глобальна змінна?
2. Що таке локальна змінна?

3. Що таке прототип функції в мові С?
4. Які в мові С є специфікатори класів пам'яті?

Задачі для домашнього завдання

Задане дійсне число *a*. Обчислити *f(a)*, якщо

1) $f(x)=\begin{cases} x^2 & \text{при} -2 \le x < 2, \\ 4 & \text{в інших випадках;} \end{cases}$

2) $f(x)=\begin{cases} 0 & \text{при } x \le 0, \\ x & \text{при } 0 < x \le 1, \\ x^4 & \text{в решті випадків;} \end{cases}$

3) $f(x)=\begin{cases} 0 & \text{при } x \le 0, \\ x^2 - x & \text{при } 0 < x \le 1, \\ x^2 \text{ - sin } \pi x^2 & \text{в решті випадків;} \end{cases}$

Написати функцію, яка реалізує основні обчислення *f(a)*, а також програму, яка зробить ввід-вивід даних і виклик функції.

Контрольні запитання

1. Які цикли називаються вкладеними?
2. За яким шаблоном будується зовнішній цикл?
3. За яким шаблоном будується внутрішній цикл?
4. Який може бути рівень вкладеності вкладених циклів?

Задачі для домашнього завдання

1. Задане натуральне число *n*. Отримати $f_1, f_2, \ldots, f_n$, де

$$f_i = \frac{1}{i^2+1} + \frac{1}{i^2+2} + \ldots + \frac{1}{i^2+i+1} \ .$$

2. Задані дійсні числа $a_1, a_2, \ldots, a_{24}$. Отримати послідовність $b_1, b_2, \ldots, b_{10}$, де

$$b_1 = a_1 + a_2 + \ldots + a_{24},$$
$$b_2 = a_1^2 + a_2^2 + \ldots + a_{24}^2,$$
$$\ldots$$
$$b_{10} = a_1^{10} + a_2^{10} + \ldots + a_{24}^{10}.$$

3. Натуральне число називається *досконалим*, якщо воно рівне сумі всіх своїх дільників, за винятком самого себе. Число 6 – досконале, тому що 6 = 1 + 2 + 3. Число 8 – не досконале, тому що 8 ≠ 1 +2 + 4.

Задане натуральне число *n*. Отримати всі доскональні числа менші за *n*.

Практичні роботи № 5, 6
Алгоритми опрацювання масивів і програмування задач, в яких використовуюся масиви.
Алгоритми опрацювання матриць і програмування задач, в яких використовуюся динамічні масиви.

Мета роботи.

Познайомити студентів з одновимірними і багатовимірними масивами. Сформувати у студентів навики написання програм на мові С з використанням масивів.

Методичні вказівки.

Опрацювання масиву полягає у виконанні операцій над його елементами. Окрім вводу-виводу масивів існує перелік найпоширеніших базових алгоритмів опрацювання масивів:

- обчислення узагальнюючих характеристик (сум, добутків і кількості елементів);
- пошук максимального чи мінімального елемента;
- пошук заданих елементів;
- перестановка елементів;
- впорядкування масивів.

Пошук заданого елемента в масиві. Найпростіший алгоритм – простий перебір (лінійний пошук). Результат пошуку – індекс елемента, який шукали або повідомлення, що такого елемента в масиві немає. Проблема – що робити, коли в масиві є декілька однакових елементів, якого шукають? Виводити індекс першого, останнього чи усіх елементів?

Функція `SearchArray` шукає в масиві `a` розміром `n` значення `x` і повертає індекс першого такого елементу, якщо ж такий елемент відсутній в масиві, то результат функції рівний -1.

```
int SearchArray(int* a, int n, int x)
{
 int i, rez=-1;
 for (i = 0; i < n; i++)
       if (x == a[i])
             return i;
 return rez;
}
```

Є й інші методи пошуку, один з них – бінарний (двійковий) пошук, який шукає елемент у відсортованому масиві. Відсортований масив за зростанням ділиться навпіл і задане значення порівнюється з центральним елементом масиву. Якщо значення, яке ми шукаємо менше за центральний елемент, то далі шукаємо в першій половині масиву, інакше в другій аналогічним способом.

Пошук максимального або мінімального елемента (або його індексу) в масиві. На початку роботи вважаємо максимальним перший елемент масиву (з індексом 0), а далі порівнюємо максимальне з іншими елементами масиву почергово. Якщо знаходимо елемент більший за максимальний, то з цього моменту максимальним вважаємо вже його.

Функція `MaxArray` шукає в масиві `a` розміром `n` найбільший елемент і повертає його в якості результату.

```
int MaxArray(int* a, int n)
```

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

Сортування масивів за зростанням або спаданням – впорядкування елементів масиву так, щоб кожен наступний був більшим (або меншим) за попередній.

Сортування методом “бульбашки” – порівнюються почергово два сусідні елементи і, якщо один з елементів не відповідає критерію сортування, то ці елементи міняються місцями.

Функція `BubbleSort` сортує масив `a` розміром `n` за зростанням.

```
void BubbleSort(int* a, int n)
{
 int i, j, temp;
 for(i = 1; i < n; i++)
       for(j=0; j < (n - i); j++)
             if (a[j] > a[j+1])
             {
                   temp = a[j];
                   a[j] = a[j + 1];
                   a[j + 1] = temp;
             }
}
```

Сортування методом вибору – знаходимо найбільший елемент і міняємо його місцями з останнім. Далі робимо аналогічну дію для масиву, не розглядаючи вже останній елемент (він уже на своєму місці).

Функція `ChoiceSort` сортує масив `a` розміром `n` за зростанням. Функція `IndexMaxArray` шукає в масиві `a` розміром `n` індекс найбільшого елементу і повертає його в якості результату.

```
int IndexMaxArray(int* a, int n)
{
 int i, max;
 max = 0;
 for (i = 1; i < n; i++)
       if (a[i] > a[max]) max = i;
 return max;
}
void ChoiceSort(int* a, int n)
{
 int i, max, temp;
 for (i = 0; i < n; i++)
 {
       max = IndexMaxArray(a, n - i);
       temp = a[max];
       a[max] = a[n - 1 - i];
       a[n - 1 - i] = temp;
 }
}
```

Сортування методом вставки – порівнюємо перший елемент масиву з усіма іншими по черзі, і коли один з елементів не відповідає критерію сортування, то цей елемент міняємо місцями з першим (вставляємо елемент на своє місце). Далі робимо аналогічну дію для масиву, починаючи з наступного елементу.

Функція `InsertSort` сортує масив `a` розміром `n` за зростанням.

```
void InsertSort(int* a, int n)
{
 int i, j, temp;
 for(i=0; i < n - 1; i++)
       for (j = i; j < n; j++)
             if (a[i] > a[j])
             {
                   temp = a[i];
                   a[i] = a[j];
                   a[j] = temp;
             }
}
```

Масиви у програмуванні широко застосовуються для зберігання і опрацювання однорідної інформації, приміром таблиць, векторів, матриць, коефіцієнтів рівнянь тощо.

Розглянемо задачу множення матриці на число і задачу множення матриці на матрицю.

Матрицею **A** розмірності *m x n* називається таблиця чисел, яка складається з *m* рядків та *n* стовпців.

$$A = A_{m\times n} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Числа, що складають матрицю, називаються її елементами і нумеруються двома індексами, які вказують на номер рядка та стовпця на перетині яких розташований даний елемент. Тобто, елемент a_{ij} міститься в *i*-му рядку та *j*-му стовпці матриці **A**.

Добутком матриці A на число k називається матриця **B** = k * **A** того ж розміру, отримана з початкової множенням на задане число всіх її елементів:

$$k * \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = \begin{pmatrix} k*a_{11} & k*a_{12} & \cdots & k*a_{1n} \\ k*a_{21} & k*a_{22} & \cdots & k*a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ k*a_{m1} & k*a_{m2} & \cdots & k*a_{mn} \end{pmatrix}$$

$$b_{ij} = k * a_{ij}$$

Функція `MultMatrixNum` множить матрицю (двовимірний масив) `a` розміром `m` рядків на `n` стовпців на число `Num`, результат отримаємо в матриці `b`.

```
void MultMatrixNum(int** a, int** b, int m, int n, int Num)
{
 int i, j;
 for (i = 0; i < m; i++)
       for (j = 0; j < n; j++)
             b[i][j] = a[i][j] * Num;
}
```

Результатом **множення матриць** $\mathbf{A}_{m\times n}$ та $\mathbf{B}_{n\times k}$ буде матриця $\mathbf{C}_{m\times k}$ така, що елемент матриці **C**, що знаходяться в і-тому рядку та j-тому стовпчику (c_{ij}), дорівнює сумі добутків елементів і-того рядку матриці **A** на відповідні елементи j-того стовпця матриці **B**:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1k} \\ b_{21} & b_{22} & \cdots & b_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ b_{n1} & b_{n2} & \cdots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ c_{m1} & c_{m2} & \cdots & c_{mk} \end{pmatrix}$$

$$c_{ij} = a_{i1} * b_{1j} + a_{i2} * b_{2j} + \cdots + a_{in} * b_{nj}$$

Дві матриці можна перемножити між собою тоді і тільки тоді, коли кількість стовпців першої матриці дорівнює кількості рядків другої матриці.

Часкові випадки – множення матриці на матрицю стовпець,

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} * \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \cdots \\ c_m \end{pmatrix}$$

$$c_i = a_{i1} * b_1 + a_{i2} * b_2 + \cdots + a_{in} * b_n$$

множення матриці рядка на матрицю

$$(a_1 \quad a_2 \quad \ldots \quad a_m) * \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix} = (c_1 \quad c_2 \quad \ldots \quad c_n)$$

$$c_i = a_1 * b_{1i} + a_2 * b_{2i} + \cdots + a_m * b_{mi}$$

і множення рядка на стовпець.

$$(a_1 \quad a_2 \quad \ldots \quad a_n) * \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{pmatrix} = (c)$$

$$c = a_i * b_i + a_i * b_i + \cdots + a_n * b_n$$

Функція `MultMatrix` множить матрицю (двовимірний масив) `a` розміром `m` рядків на `n` стовпців на матрицю `b`, розміром `n` рядків на `k` стовпців, результат отримаємо в матриці `c`. Функція працює і для часткових випадків, в такому випадку для матриці рядка кількість стовпців необхідно задавати рівним 1, а для матриці стовпця кількість рядків рівною 1.

```
void MultMatrix(int** a, int** b, int** c, int m, int n, int k)
{
 int i, j, l;
 for (i = 0;i < m;i++)
       for (j = 0;j < k;j++)
       {
             c[i][j] = 0;
             for (l = 0; l < n; l++)
                   c[i][j] = c[i][j] + a[i][l] * b[l][j];
       }
}
```

Для роботи з матрицями утворимо динамічний двовимірний масив розміром `m` рядків і `n` стовпців функцією `CreateMatrix`, яка поверне вказівник на початок масиву:

```
int** CreateMatrix(int m, int n)
{
 int i;
```

```
 int** a;
 a = new int* [m];
 for (i = 0;i < m;i++)
       a[i] = new int[n];
 return a;
}
```

Звільнити виділену пам'ять під масив `a` розміром `m` рядків і `n` стовпців можна функцією `DeleteMatrix`:

```
void DeleteMatrix(int** a, int m, int n)
{
 int i;
 for (i = 0;i < m;i++)
       delete[]a[i];
 delete[]a;

}
```

Функція `FillMatrix` заповнює масив `a` розміром `m` рядків і `n` стовпців випадковими числами (в діапазоні від 0 до 10):

```
void FillMatrix(int** a, int m, int n)
{
 int i, j;
 for (i = 0;i < m;i++)
       for (j = 0;j < n;j++)
             a[i][j] = rand() % 10;
}
```

Функція `PrintMatrix` виводить на екран елементи масиву `a` розміром `m` рядків і `n` стовпців:

```
void PrintMatrix(int** a, int m, int n)
{
 int i, j;
 for (i = 0;i < m;i++)
 {
       printf("\n");
       for (j = 0;j < n;j++)
             printf("%7d", a[i][j]);
 }
 printf("\n");
}
```

Контрольні запитання

1. Як оголосити одновимірний масив в мові програмування С?
2. Як оголосити багатовимірний масив в мові програмування С?
3. Як виділити пам'ять під динамічний масив?
4. Як передати масив в якості параметра функцій?

Задачі для домашнього завдання.

1. Написати програму, яка порівняє різні методи сортування масивів за кількістю перестановок.
2. Написати функцію пошуку кількості максимальних значень у масиві.
3. Написати функцію, яка перемножить матрицю рядок на матрицю стовпець.

Практична робота № 7
Програмування на С задач, в яких використовуються рядки.

Мета роботи.

Познайомити студентів з рядками. Сформувати у студентів навики написання програм на мові С з використанням рядків.

Методичні вказівки

Рядки у С являють собою послідовність (масив) символів із завершальним нуль-символом. Нуль-символ (нуль-термінатор) – це символ з кодом 0, який записується у вигляді керуючої послідовності '\0'. За розташуванням нуль-символу визначається фактична довжина рядка.

Рядок може бути оголошеним в один з нижче наведених способів:

1) `char *s; //` Оголошення вказівника на перший символ рядка;

`//` *пам'ять під сам рядок не виділяється*

2) `char ss[15]; //` Оголошення рядка ss з 14-ти символів;

`//` *пам'ять виділяється компілятором*

3)
```
const int n = 10;
char st[n]; // Оголошення рядка st з n-1 (тобто 9-ти) символів;

// пам'ять виділяється компілятором
```

4)
```
int n = 10;
char *str = new char[n]; // Оголошення рядка str з n-1 (тобто 9)

// символів; пам'ять виділяється динамічно
```

При зазначенні довжини рядка слід враховувати завершальний нуль-символ. Наприклад, у вищенаведеному рядку str можна зберігати не 10, а лише 9 символів.

Зауважимо, що при оголошенні рядка першим способом пам'ять під рядок не виділяється і це може бути дуже небезпечним, оскільки до тієї самої ділянки пам'яті може бути розміщено інші змінні й рядок буде втрачено.

При оголошенні рядок можна ініціалізувати рядковою константою, при цьому нуль-символ формується автоматично після останнього символу:

```
char str[10] = "Vasia";
```

При цьому виділиться пам'ять під масив з 9-ти елементів та 10-й – нуль-символ (всього 10 байт) і перші 5 символів рядка записуються в перші 5 байт цієї пам'яті (`str[0]='V'`, `str[1]='a'`, `str[2]='s'`, `str[3]='i'`, `str[4]='a'`), а в шостий елемент `str[5]` записується нуль-символ. Якщо рядок при оголошенні ініціалізується, його розмірність можна опускати (компілятор сам виділить потрібну кількість байтів):

```
char str[] = "Vasia"; // Виділено й заповнено 6 байтів
```

Контрольні запитання

1. Чим масив символів відрізняється від рядка в С?
2. Що таке рядок в мові С?
3. Які є функції для вводу рядків?

4. Які є функції для виводу рядків?
5. Що таке довжина рядка?
6. Яким чином визначається фактична довжина рядка?

Задачі для домашнього завдання

1. Ввести з клавіатури рядок і вивести його посимвольно на екран.
2. Ввести рядок і визначити чи є у ньому цифри.

Практична робота № 8
Програмування на С задач, які використовують текстові та бінарні файли для вводу та виводу даних.

Мета роботи.

Познайомити студентів з текстовими та бінарними файлами, функціями для роботи з ними в мові С. Сформувати у студентів навики написання програм на мові С з використанням файлів.

Методичні вказівки

С надає засоби опрацювання двох типів файлів: *текстових* та *бінарних*.

Текстові файли призначено для зберігання текстів, тобто сукупності сим-вольних рядків змінної довжини. Кожен рядок завершується керуючою послідовністю '\n', а розділювачами слів та чисел у рядку є пробіли й символи табуляції. Оскільки вся інформація текстового файлу є символьною, програмне опрацювання такого файлу полягає в читанні рядків, виокремленні з рядка слів і, за потреби, перетворюванні цифрових символьних послідовностей на числа відповідними функціями перетворювання. Створювати, редагувати текстові файли можна не лише в програмі, а й у якому завгодно текстовому редакторі, наприклад Блокноті чи Word.

Бінарні файли зберігають дані в тому самому форматі, в якому вони були оголошені, і їхній вигляд є такий самий, як і в пам'яті комп'ютера. І тому відпадає потреба у використанні розділювачів: пробілів, керуючих послідовностей, а отже, обсяг використовуваної пам'яті порівняно з текстовими файлами з аналогічною інформацією є значно меншим. Окрім того, немає потреби у застосуванні функцій перетворення числових даних. Але кожне опрацювання даних бінарних файлів можливе лише за наявності програми, якій має бути відомо, що саме і в якій послідовності зберігається у цьому файлі.

Файлова змінна – вказівник потоку, який в подальшому буде передаватися у функції введення-виведення у якості параметра:

```
FILE *f;
```

Функція `fopen()` для відкривання файлу має такий синтаксис:

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

Перший параметр `filename` визначає ім'я файлу, який відкривається. Другий параметр `mode` задає режим відкривання файлу.

Специфікатори режиму відкривання файлів

Параметр	Опис
r	відкрити файл лише для зчитування даних
r+	відкрити існуючий файл для зчитування й записування даних
a	відкрити чи створити файл для записування даних у кінець файла
a+	відкрити чи створити файл для зчитування чи то записування даних у кінець файла
w	створити файл для записування даних
w+	створити файл для зчитування і записування даних

До зазначених специфікаторів наприкінці чи перед знаком "+" може до-писуватись символ чи то "t" – для текстових файлів, чи "b" – для бінарних (двійкових) файлів.

Функція `fopen()` повертає вказівник на об'єкт, який керує потоком. Наприклад, створити файл з ім'ям `Prim.txt` можна так:

```
f = fopen("Prim.txt", "wt");
```

Якщо файл відкрити не вдалося, `fopen()` повертає нульовий вказівник `NULL`. Для уникання помилок після відкриття файла слід перевірити, чи насправді файл відкрився:

```
if(f == NULL) {ShowMessage("Файл не відкрито"); return;}
```

Припинити роботу з файлом можна за допомогою функції
`fclose``(FILE *f).`

Ця функція закриває файл, на який посилається параметр функції.

Наведемо приклад відкривання текстового файла для зчитування.

```
FILE *f;

// Перевіряємо, чи повертає ця функція нульовий вказівник
if((f=fopen("t.txt","rt"))==NULL)

{ ShowMessage("Неможливо відкрити файл"); return; }
. . . // Сюди вставляються команди зчитування з файла.

fclose(f); // Закриття файла.
```

З текстового файла можна читати цілі рядки й окремі символи.

Зчитування рядка з файла здійснюється функцією `fgets()`:

```
char *fgets(char *s, int m, FILE *stream);
```

де `s` – перший параметр – рядок типу `char*`;

`m` – кількість читаних символів (байтів);

`stream` – вказівник на потік даних файла.
Перевірка кінця файла здійснюється функцією `feof()`:

```
int feof (FILE *stream);
```

Записування даних до текстового файла можна здійснювати за допомогою функції

```
int fputs(const char *s, FILE *stream);
```

де `s` – рядок типу `char`,

`stream` – файловий потік.

Для записування даних до файла слід відкрити файл у форматі записування даних у кінець файла, за потреби перетворити рядок на тип `char` і записати дані до файла.

Перевірка кінця файла здійснюється функцією `feof()`:

```
int feof (FILE *stream);
```

Текстові файли дозволяють переміщувати поточну позицію зчитування-запису. Для визначення поточної позиції файла, яка автоматично зміщується на кількість опрацьованих байт, використовується функція `ftell()`:

```
long int ftell(FILE *stream);
```

А змінити поточну позицію файла можна за допомогою функції `fseek()`:

```
int fseek(FILE *stream, long offset, int whence);
```

Ця функція задає зсув на кількість байтів `offset` щодо точки відліку, яка зада-ється параметром `whence`. Параметр `whence` може набувати таких значень:

0 – початок файлу,

1 – поточна позиція,

2 – кінець файлу.

Наприклад, для переміщення поточної позиції на початок файла можна скористатися функцією

```
fseek(f, 0, 0);
```

За допомогою функцій `ftell()` та `fseek()` можна визначити сумарний обсяг пам'яті у байтах, який займає файл. Для цього є достатньо переміститися на кінець файла:

```
fseek(f, 0, 0);
int d = ftell(f);
```

У *бінарному* (двійковому) файлі число, на відміну від текстового, зберігається у внутрішньому його поданні. У двійковому форматі можна зберігати не лише числа, а й рядки та цілі інформаційні структури. Причому останні зберіга-ти зручніше, завдяки тому що відсутня потреба явно зазначати кожен елемент структури, а зберігається вся структура як цілковита одиниця даних. Хоча цю інформацію не можна прочитати як текст, вона зберігається більш компактно і точно. Тому, що саме і в якій послідовності розміщено в бінарному файлі, має бути відомо програмі.

Функція `fread()` читає інформацію у вигляді потоку байтів і в незмінному вигляді розміщує її в пам'яті. Слід розрізнювати текстове подавання чисел і їхнє бінарне подавання.

```
size_t fread
( char *buffer, // Масив для зчитування даних,
size_t elemSize, // розмір одного елемента,
size_t numElems, // кількість елементів для зчитування
FILE *f // і вказівник потоку
);
```

Тут `size_t` означений як беззнаковий цілий тип у системних заголовних файлах. Функція намагається прочитати `numElems` елементів з файла, який задаєть-ся вказівником `f`, розмір кожного елементу становить `elemSize`. Функція пове-ртає реальну кількість прочитаних елементів, яка може бути менше за `numElems`, у разі завершення файла чи то помилки зчитування.

Функція бінарного записування до файла `fwrite()` є аналогічна до функції зчитування `fread()`. Вона має такий прототип:

```
size_t fwrite

( char *buffer, // Масив записуваних даних,

size_t elemSize, // розмір одного елемента,

size_t numElems, // кількість записуваних елементів

FILE *f // і вказівник потоку
);
```

Функція повертає кількість реально записаних елементів, яка може бути менше за `numElems`, якщо при записуванні виникла помилка: приміром, не вистачило вільного простору на диску.

Так само, як і в текстових, у бінарних файлах можна визначати позицію зчитування-записування і переміщувати її у довільне місце файла командами відповідно `ftell()` та `fseek()`. Можливість переміщувати позицію є корисна для файлів, які складаються з

однорідних записів однакового розміру. Приміром, якщо у файлі записано лише дійсні числа типу `double`, то для того, щоб прочитати `i`-те число, слід виконати оператори:

```
fseek(F, sizeof(double)*(i-1), 0);
fread(&a, sizeof(double), 1, F);
```

Контрольні запитання

1. Що таке текстовий файл?
2. Що таке бінарний файл?
3. Які є специфікатори режимів відкриття файлів?
4. Які є функції запису-читання з текстового файлу?
5. Які є функції запису-читання з бінарного файлу?

Задачі для домашнього завдання

1. Записати у текстовий файл своє ім’я, по-батькові, прізвище, групу.
2. Записати у бінарний файл наступну інформацію – прізвище, групу, дату народження.