МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Методичні рекомендації
до виконання практичних завдань
з навчальної дисципліни

"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

для студентів напряму підготовки
"Комп'ютерні науки"
всіх форм навчання

Харків. Вид. ХНЕУ, 2010

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 4 від 08.01.2010 р.

М54 Методичні рекомендації до виконання практичних завдань з навчальної дисципліни "Основи програмування та алгоритмічні мови" для студентів напряму підготовки "Комп'ютерні науки" всіх форм навчання / укл. О. В. Тарасов, В. М. Федорченко, М. Ю. Лосєв. – Харків : Вид. ХНЕУ, 2010. – 92 с. (Укр. мов.)

Викладено матеріал, що включає методику проведення практичних занять з даної навчальної дисципліни. Велика увага приділяється засвоєнню методу використання теорії, придбання професійних вмінь (компетенцій), а також практичних вмінь, необхідних для виконання лабораторних робіт. Наведено багато практичних прикладів, що ілюструють реалізацію конкретних завдань.

Рекомендовано для студентів, викладачів і користувачів, які вивчають основи програмування на алгоритмічній мові С++.

Методичні рекомендації

Практичне заняття — це форма організації навчального процесу, що припускає виконання студентами за завданням і під керівництвом викладача однієї або декількох практичних робіт. І якщо на лекції головна увага студентів зосереджується на роз'ясненні теорії навчальної дисципліни, то практичні заняття служать для навчання методам її застосування. Як правило, практичні заняття проводяться паралельно із читанням усіх основних навчальних дисциплін. Головною метою практичних занять є засвоєння методу використання теорії, придбання професійних вмінь (компетенцій), а також практичних вмінь, необхідних для виконання лабораторних робіт (ІНДЗ) і вивчення наступних дисциплін.

Практичні заняття відіграють важливу роль у виробленні у студентів навичок застосування отриманих знань для вирішення практичних завдань разом з викладачем. Практичні заняття з навчальної дисципліни проводяться в комп’ютерних класах через 2 лекції і логічно продовжують роботу, почату на лекції, допомагають студенту якісно виконати завдання поточної лабораторної роботи.

Загальна структура практичних занять: вступне слово викладача; відповіді на запитання студентів; практична частина як планова; заключне слово викладача.

Найважливішою частиною будь-якої форми практичних занять є вправи. Основа вправи — приклад, який розбирається з позицій теорії, розвинутої в лекції. Як правило, основна увага приділяється формуванню конкретних вмінь та навичок, що визначає зміст діяльності студентів. Проводячи вправи, слід спеціально звертати увагу на формування здатності до алгоритмічного мислення, осмислення і розуміння задачі. Ви повинні розкрити і виявити свої здатності, свій особистий потенціал.

Критерії ефективності практичного заняття: цілеспрямованість, планування, організація, стиль проведення, відносини "викладач — студенти", керування групою.

Усі практичні заняття з навчальної дисципліни проводяться за єдиною схемою:

Вступ (5 — 8 хвилин): перевірка відвідування занять, оголошення теми, мети, основних питань і порядку проведення заняття; проведення

контрольного опитування, де студенти письмово відповідають на запитання з матеріалу поточних лекцій і здають відповіді.

Постановка й аналіз задачі (7 — 8 хвилин): студенти розбирають фізичну сутність, вхідні та вихідні дані, виконують математичний опис задачі, що пропонується викладачем. Один зі студентів працює біля дошки.

Створення специфікації програми (7 — 8 хвилин): студенти створюють специфікацію програми та вибирають форму вхідних/вихідних даних для задачі, що запропонована викладачем. Один зі студентів виконує завдання біля дошки.

Побудова алгоритму (7 — 8 хвилин): на підставі математичного опису задачі студенти розробляють порядок дій та схему алгоритму розглянутої задачі. Один зі студентів виконує завдання біля дошки.

Розробка програми, трансляція і редагування програми, тестування програми (15 — 25 хвилин): кожен студент самостійно створює програмний код (додаток), для рішення задачі використовує приклад, наведений у методичному посібнику, у ході компіляції програми виявляються та виправляються синтаксичні помилки, у ході тестування виявляються логічні помилки. Викладач відповідає на запитання, індивідуально допомагає студентом.

Самостійна робота (25 — 30 хвилин): студенти самостійно або в мінігрупах розв’язують задачу, що запропонована викладачем, викладач відповідає на запитання, індивідуально допомагає, оцінює роботу студентів.

Закінчення (5 хвилин): підсумки заняття, аналіз роботи групи на занятті, узагальнення питань, які частіше всього виникали при виконанні завдань заняття. Отримання завдання для самопідготовки.

Практичні заняття покликані для поглиблення, розширення, деталізації знань, отриманих на лекції в узагальненій формі, і сприяють виробленню навичок професійної діяльності. Вони розбудовують наукове мислення й мову, дозволяють перевірити знання студентів і виступають як засіб оперативного зворотного зв'язку.

Практичне заняття не повинне бути "топтанням на одному місці". Слід організувати свою роботу на практичному занятті так, щоб постійно відчувати зростання складності виконуваних завдань, отримувати позитивні емоції від переживання власного успіху у програмуванні, бути

зайнятим напруженою творчою роботою, пошуками правильних і точних розв'язків.

1. Розробка графічних схем алгоритмів

1.1. Алгоритм як основне поняття програмування

Термін "алгоритм" пов'язаний з ім'ям одного з найбільш відомих вчених середньовічної Середньої Азії Мухамеда ібн Мусі аль-Хорезмі (783 – 850 рр.). Теорія алгоритмів має велике практичне значення. Алгоритмічний тип діяльності людини важливий як один з могутніх та ефективних форм її праці. Через алгоритмізацію, через розбиття складних дій на простіші, на дії, виконання яких доступне ЕОМ, пролягає шлях до автоматизації.

Таким чином, а*лгоритм* є певною послідовністю дій, які необхідно виконати, щоб отримати результат. Алгоритм може описувати собою деяку послідовність обчислень, а може — послідовність дій нематематичного характеру. Іншими словами, а*лгоритм* — це зрозуміле і точне розпорядження для здійснення певної послідовності дій, спрямованих на рішення конкретної задачі. Якщо алгоритм призначений для реалізації на ЕОМ, то він перетвориться у спеціальну програму, на одному або декількох алгоритмічних мовах, доступних на сучасних обчислювальних машинах.

Для будь-якого алгоритму справедливі загальні закономірності — властивості алгоритму:

1. Дискретність.
2. Зрозумілість.
3. Детермінованість.
4. Масовість.
5. Результативність.

Дискретність — це властивість алгоритму, що дозволяє розбити його на кінцеву кількість елементарних дій (кроків).

Зрозумілість — властивість алгоритму, при якій кожен з цих елементарних дій (кроків) є закінченими і зрозумілими.

Детермінованість — це властивість, коли кожна дія (операція, вказівка, крок, вимога) повинна розумітися у строго певному сенсі, щоб не залишалося місця довільному тлумаченню, щоб кожен, хто прочитав

вказівку, розумів його однозначно.

Масовість — властивість, коли за даним алгоритмом повинна вирішуватися не одна, а цілий клас подібних задач.

Результативність — властивість, при якій будь-який алгоритм у процесі виконання повинен приводити до певного результату. Негативний результат також є результатом.

Для представлення (опису) алгоритму використовуються різні способи:

1. Вербальний (словесний) опис;
2. Графічна схема алгоритму;
3. Таблиця рішень;
4. Опис засобами алгоритмічної мови програмування.

Прикладів таких записів безліч. Наприклад книга кулінарних рецептів є ні чим іншим, як збіркою алгоритмів, написаних рідною мовою. Проте при програмуванні алгоритмів, з метою їх переведення на ЕОМ, найбільш прийнятним і часто використовуваним, є подання алгоритму у вигляді блок-схеми (або просто схеми).

1.2. Правила оформлення блок-схем

При створенні блок-схем необхідно користуватися правилами, затвердженими ГОСТ 19.701-90 (Міжнародний стандарт ІСО 5807-85). Нижче наводяться витяги з цього нормативного документа.

1.2.1. Загальні положення

Схеми алгоритмів, програм, даних і систем (далі — схеми) складаються з конкретних символів, що мають певне значення (часто званих блоками), короткого тексту пояснення і сполучаючих ліній. Усі символи (блоки) алгоритму повинні бути пронумеровані.

Схеми можуть використовуватися на різних рівнях деталізації, причому кількість рівнів залежить від розмірів і складності завдання обробки даних. Рівень деталізації повинен бути таким, щоб різні частини

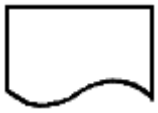
і взаємозв'язок між ними були зрозумілі в цілому.

1.2.2. Опис символів

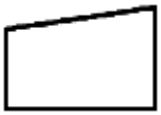
1.2.2.1. Символи даних

Дані. Символ відображає дані, носій даних не визначений. Ця фігура часто використовується і для запису різних алгоритмів, тому як для алгоритму джерело інформації не істотне і визначається він тільки при реалізації програми.

Документ. Символ відображає дані, представлені на носієві в легкій для читання формі (машинограма, документ для оптичного або магнітного прочитування, мікрофільм, рулон стрічки з підсумковими даними, бланки введення даних).



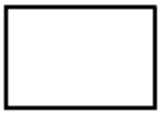
Ручне введення. Символ відображає дані, що вводяться вручну під час обробки з пристроїв будь-якого типу (клавіатура, перемикачі, кнопки, світлове перо, смужки з штриховим кодом).



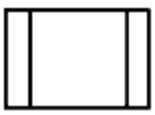
Дисплей. Символ відображає дані, представлені у формі, що легко читається людиною на спеціальному пристрої, який може бути екраном для візуального спостереження, індикатором введення інформації тощо.

1.2.2.2. Символи процесу

Процес. Символ відображає функцію обробки даних будь-якого вигляду (виконання певної операції або групи операцій, що приводить до зміни значення, форми або розміщення інформації або до визначення, за яким з декількох напрямів потоку слід рухатися).



Зумовлений процес. Символ відображає зумовлений процес, що складається з однієї або декількох операцій чи кроків програми, які визначені в іншому місці (у підпрограмі,



модулі). Тобто цей елемент використовується для позначення підпрограм.

Підготовка. Символ відображає модифікацію команди або групи

команд з метою впливу на деяку подальшу функцію (установка перемикача, модифікація індексного регістра або ініціалізація програми). Часто використовується для завдання параметрів рахункового оператора циклу (див. приклад блок-схеми алгоритму рис. 1.3).

Рішення. Символ відображає рішення або функцію типу перемикача, що має один вхід і ряд альтернативних виходів, один і лише

один з яких може бути активізований після обчислення умов, визначених усередині цього символу. Відповідні результати обчислення можуть бути записані по сусідству з лініями, що відображають ці шляхи. Використовується для позначення оператора **IF** або меж циклів з пост- і передумовами.

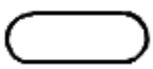
1.2.2.3. Спеціальні символи

З'єднувач. Символ відображає вихід у частину схеми і вхід з іншої

частини цієї схеми та використовується для обриву лінії і продовження її в іншому місці. Відповідні символи-з'єднувачі повинні містити одне і те ж унікальне позначення.

Часто використовується при розміщенні блок-схеми на декількох листах. Межа на одному листі позначається літерою **A**, а на іншому листі починається із з'єднувача, в якому також присутня літера **A**. При розміщенні на трьох листах, в кінці другого листа поміщаємо з'єднувач з літерою **B** і т. д.

Термінатор. Символ відображає вихід в зовнішнє середовище і



вхід із зовнішнього середовища (початок або кінець схеми програми, зовнішнє використання і джерело або пункт призначення даних).

Коментар. Символ використовують для додавання описових коментарів або записів пояснень у цілях пояснення або приміток. Пунктирні лінії в символі коментаря пов'язані з відповідним символом

або можуть обводити групу символів. Текст коментарів або приміток повинен бути поміщений біля обмежуючої фігури.

Приклад використання коментаря у схемах наведений на рис 1.1.

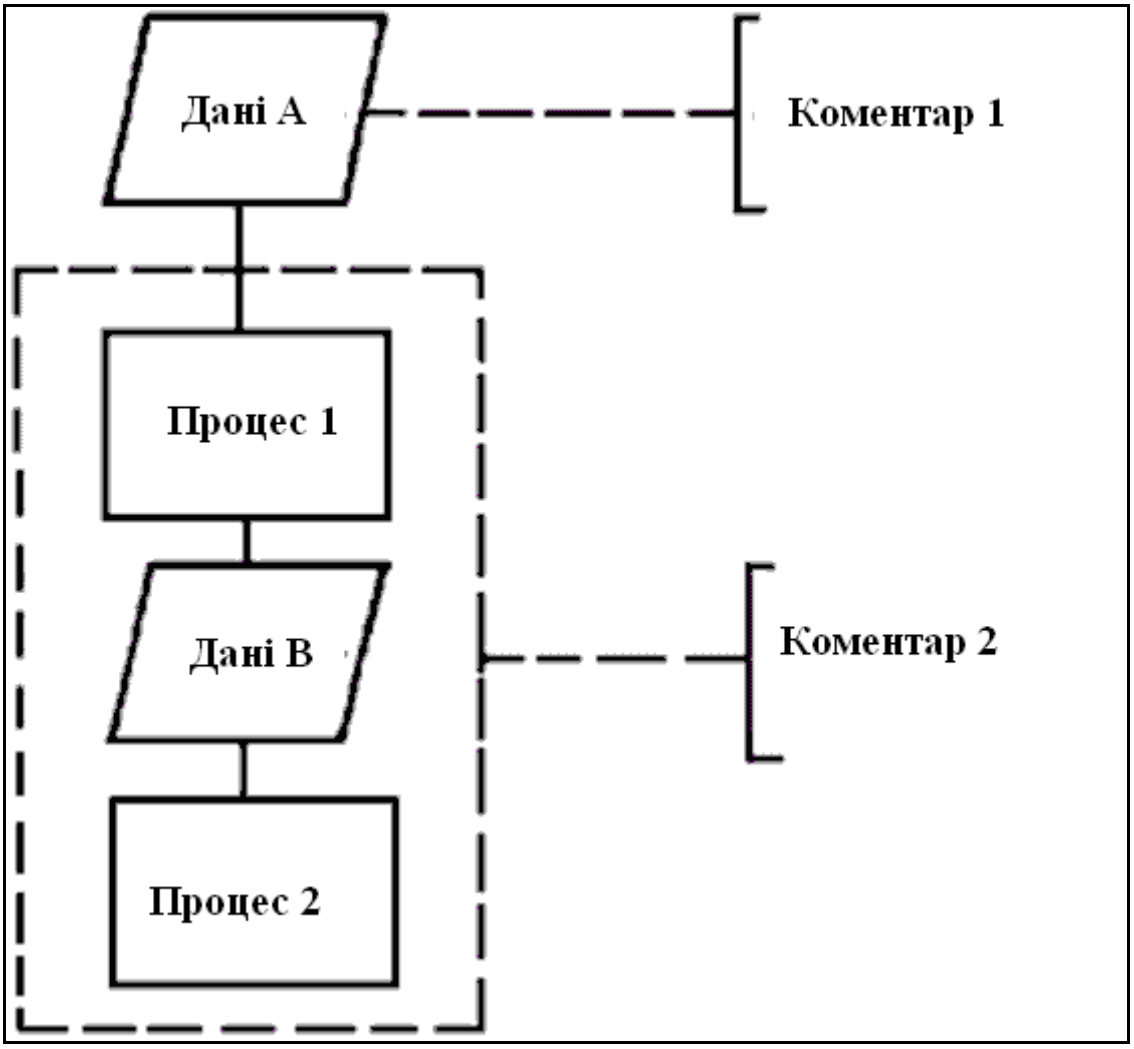


Рис. 1.1. **Приклад блок-схеми програми із специфікації ГОСТ 19.701-90**

1.3. Типи алгоритмів

Алгоритми бувають трьох типів:

послідовний — дії виконуються по порядку один за одним (рис.1.2);

циклічний — організовує повторення дій (рис.1.3);

такий, що розгалужується — містить одне або декілька логічних умов і має декілька гілок обробки (рис.1.4). Розгалуження дає можливість

вибору варіанта дій залежно від результатів аналізу початкових умов.

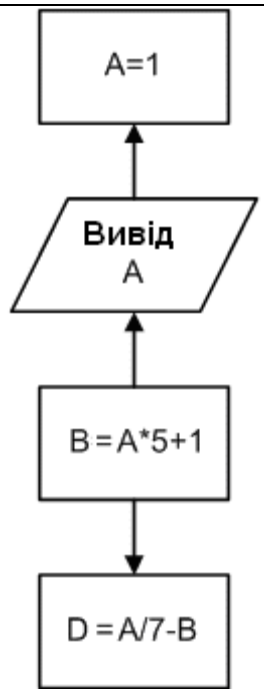
Послідовний алгоритм	
	A=1; printf("A=%d \n",A); B=A*5+1; D=A/7-B;

Рис.1.2. **Приклад послідовного алгоритму**

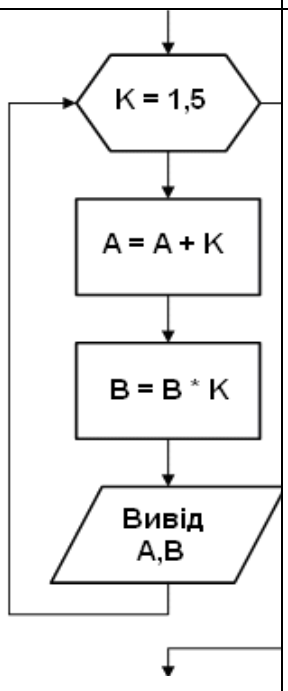
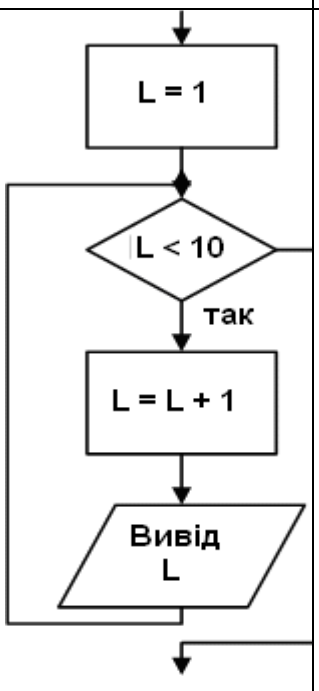
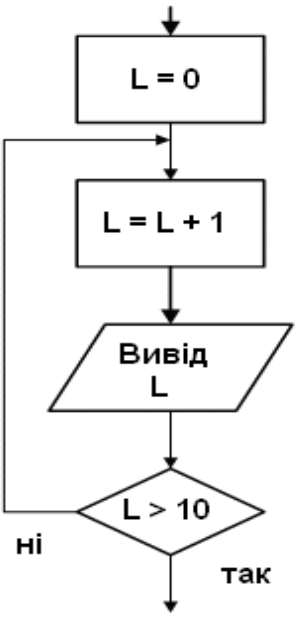
Циклічні алгоритми		
Оператор циклу **FOR**	Умовний оператор циклу з передумовою **WHILE**	Умовний оператор циклу з передумовою **WHILE**
		

Рис.1.3. **Приклад циклічних алгоритмів**

for (int K=1; K<=5; K++) { **A=A+K;** **B=B*K;** **cout<<A<<B<<"\n";** **}**	**L=1;** **while(L<10) {** **L++;** **printf("L=%d \n",L);** **}**	**L=0;** **do {** **L++;** **printf("L=%d \n",L);** **} while (L<10);**

Закінчення рис.1.3.

Алгоритми, що розгалужуються		
Умовний оператор **IF** (повний запис)	Умовний оператор **IF** (короткий запис)	Оператор вибору **SWITCH-CASE**

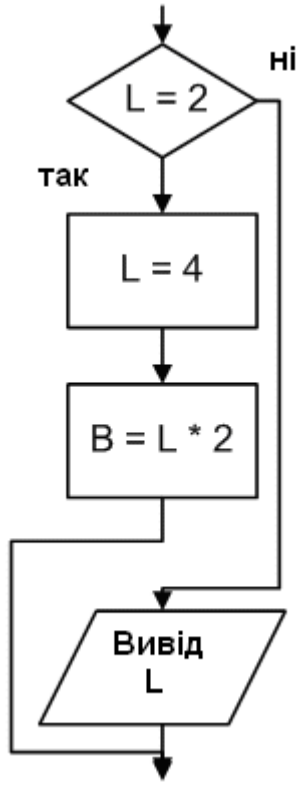
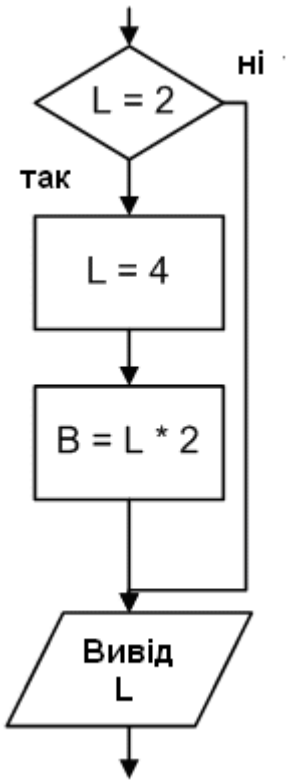
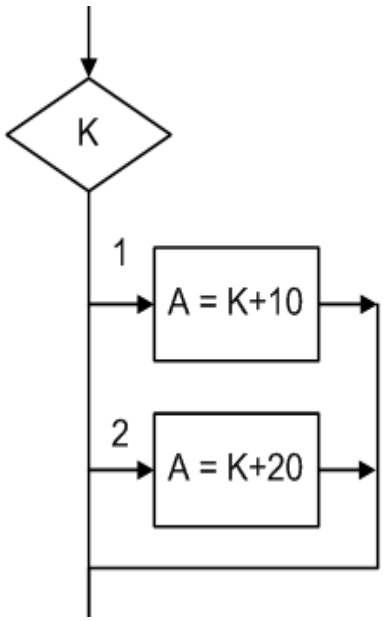
		
if (L==2) { **L=4;** **B=L*2;** **} else {** **printf("L=%d \n",L);** **}**	**if (L==2) {** **L=4;** **B=L*2;** **}** **printf("L=%d \n",L);**	**switch (K) {** **case 1: A=K+10; break;** **case 2: A=K+20; break;** **……..** **}**

Рис.1.4. **Приклад алгоритмів, що розгалужуються**

Розглянемо тепер два приклади, в яких використовується як лінійні ділянки, так і розгалуження та цикли.

Приклад 1.1. Побудувати блок-схему алгоритму рішення квадратного рівняння Ax2+bx+c=0 (рис.1.5).

Проаналізуйте дану схему і вкажіть, при виконанні яких умов, виконання алгоритму повинне йти по тій або іншій гілці.

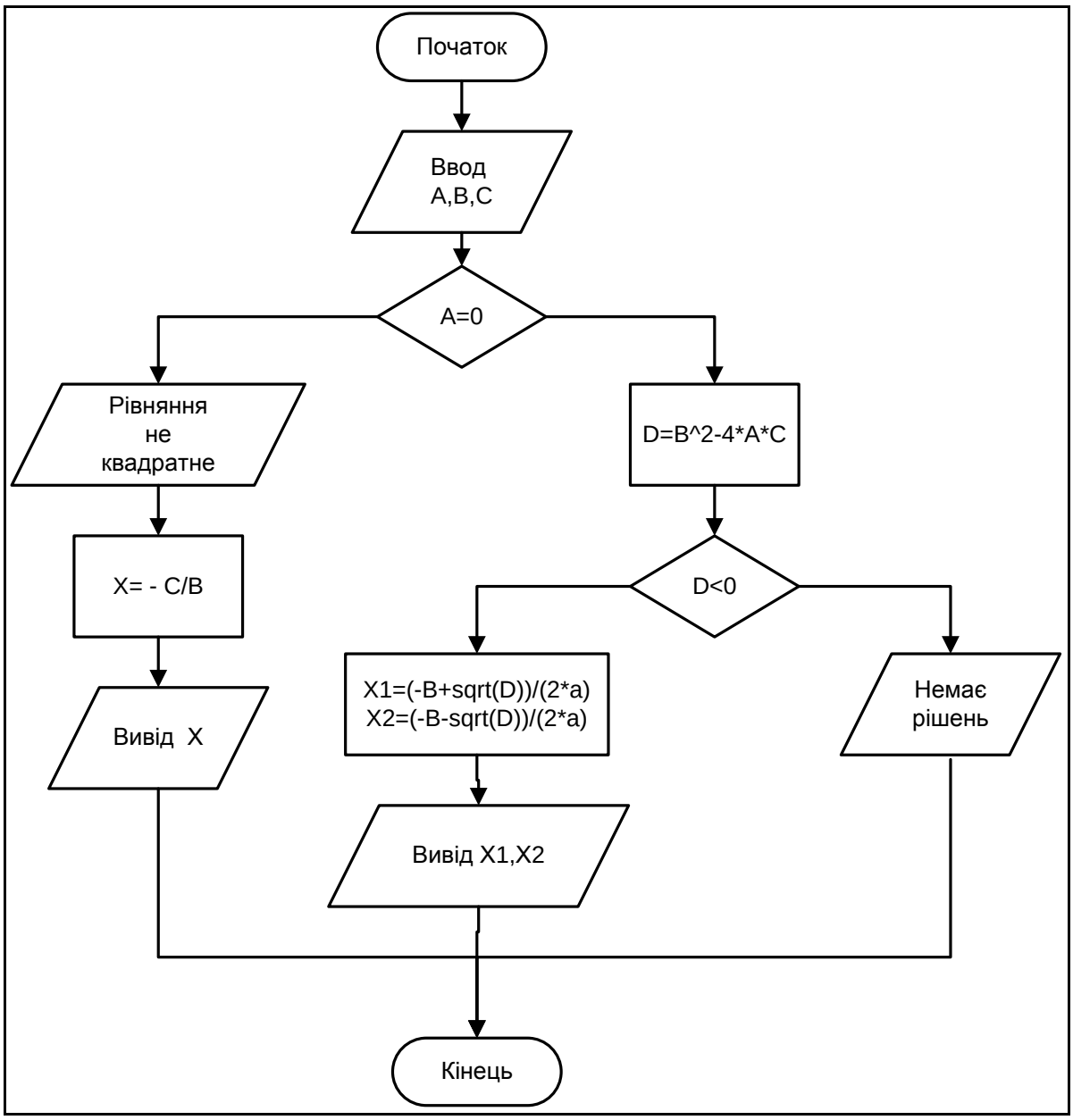


Рис.1.5. **Блок-схема алгоритму рішення квадратного рівняння**

Приклад 1.2. Побудувати блок-схему розрахунку значень функції f(x) на заданому інтервалі [a,b] з кроком dx.

Припустимо, що функція виражена таким аналітичним виразом:

$$f(x)=\begin{cases}(x+1)/(x+2), & \text{при } x<0\\ (x+0.5)/x, & \text{при } x>=0\end{cases}$$

Блок-схема алгоритму поставленого завдання (один з можливих варіантів) виглядатиме таким чином (рис.1.6).

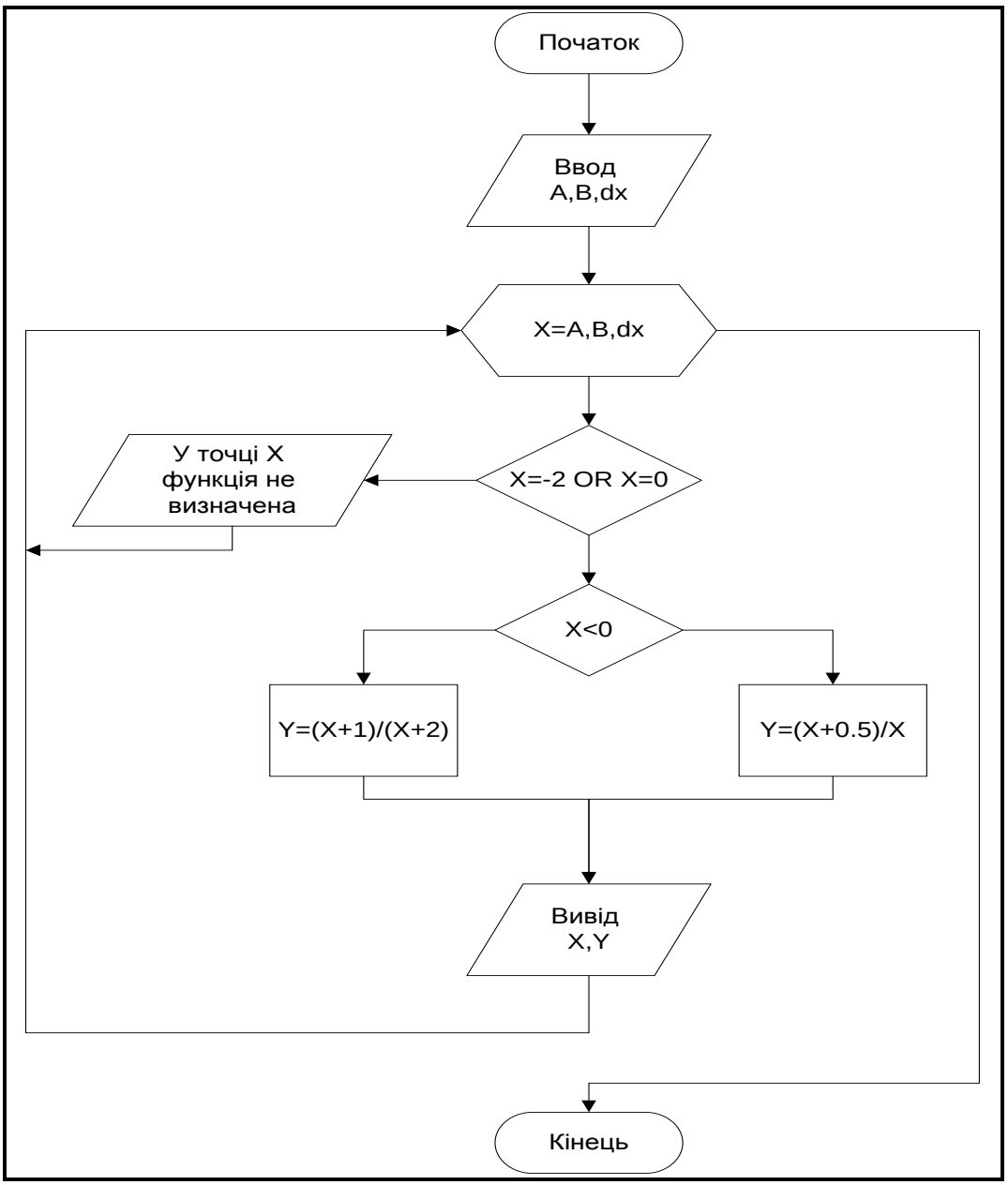


Рис. 1.6. **Блок-схема алгоритму розрахунку значень функції f(x) на заданому інтервалі [a,b] з кроком dx**

У блок-схемах алгоритмів, наведених на рис.1.5 та 1.6, визначте, при виконанні яких умов, будуть виконуватися ті чи інші гілки алгоритму.

2. Розробка лінійних програм

Лінійні програми відносяться до класу найбільш елементарних програм. Алгоритм вирішення таких програм не припускає ніяких перевірок умов, складних циклів і тому подібного і зазвичай складається з трьох основних кроків:

1. Введення початкових даних (дані можуть задаватися як безпосередньо в програмі, так і вводиться в неї із зовнішнього джерела, наприклад клавіатури або файла).

2. Обчислення за заздалегідь заданими формулами значень певних показників.

3. Виведення отриманих результатів (на екран, у файл, на принтер і тому подібне). Розбираючи приклади подібних програм, слід звернути особливу увагу на правила запису виразів у мові програмування, оскільки вони відрізняються від запису формул у математиці.

Основний теоретичний матеріал, що підлягає опрацьовуванню перед написанням програм, полягає в такому:

загальна структура програм на мові С++;

призначення і використання заголовних файлів;

основні арифметичні операції і правила запису арифметичних дужкових і бездужкових виразів;

основні стандартні математичні функції;

способи введення і виведення даних у програму.

Завдання 2.1. Розрахувати значення параметра A, заданого такою формулою.

$$A = \frac{x^{y+1} + e^{2y-1}}{3 + x\left|y - \operatorname{tg} z\right|}.$$

Значення X задати безпосередньо у програмі, а значення Y та Z — ввести з клавіатури.

```
#include "stdafx.h"
#include <locale.h>
#include <iostream>
#include <math.h>

using namespace std;
```

```
int main()
{
   // Визначення змінних, що беруть участь в програмі
   // Змінна x відразу ініціалізіруєся значенням рівним 1
   double x=1,y,z,A;

   // Установка виводу на екран символів кирилиці
   setlocale(0,"RUS");

   // Вивід на екран заголовка програми
   cout <<"*****************************************\n";
   cout <<" Програма визначення значення параметра A \n";
   cout <<"      як функції від x, y и z\n";
   cout <<"*****************************************\n";
   cout <<"Значение X задано равным "<< x << "\n";

   // Введення значень змінних y і z з клавіатури
   cout <<" Введіть значення Y= "; cin >> y;
   cout <<" Введіть значення Z= "; cin >> z;

   // Розрахунок значення параметра A по заданій формулі
   A=(pow(x,y+1)+exp(2*y-1))/(3+x*fabs(atan(z)));

   // Виведення отриманог значення на екран
   cout <<" Отримане значення A="<<A<<"\n\n";

   // завершення програми з цілим кодом
   // оскільки функція main має тип int
   return 0;
}
```

Завдання 2.2. Ввести з клавіатури значення сторін паралелепіпеда (a, b, c) і визначити: площу основи; об'єм фігури; довжину діагоналі.

```
#include "stdafx.h"
#include <locale.h>
#include <stdio.h>
```

```
#include <math.h>

int main()
{  // Установка виводу на екран символів кирилиці
   setlocale(0,"RUS");
   float a,b,c;  // визначення змінних для зберігання довжини сторін фігури
   double S,V,D; // визначення змінних для площі, об'єму і довжини діагоналі

   // Виведення інформації про програму
   printf("Програма визначення параметрів паралелепіпеда \n");
   printf("   по значенню довжини трьох його сторін \n\n");
   // Введення початкових даних
   printf("Введіть значення сторін паралелепіпеда:\n");
   printf("a="); scanf("%f",&a); // введення довжини a
   printf("b="); scanf("%f",&b); // введення довжини b
   printf("c="); scanf("%f",&c); // введення довжини c
   // Виведення введених результатів для візуального контролю
   printf("Введені наступні значення:\n a=%6.3f b=%6.3f c=%6.3f\n\n",a,b,c);

   S=a*b;              // Визначення площі основи
   V=S*c;              // Визначення об'єму
   D=sqrt(a*a+b*b+c*c);   // Знаходження довжини діагоналі

   // Виведення знайдених значень на екран
   printf("Площа основи =%8.3lf\n",S);
   printf("Об'єм        =%8.3lf\n",V);
   printf("Діагональ    =%8.3lf\n",D);
   return 0;
}
```

3. Розробка розгалуджених програм

Для успішного освоєння даної теми необхідно детально опрацювати теоретичний матеріал і особливо приділити увагу:

основним операторам перевірки умови у програмах;

операторові вибору і правилам його використання;

способам заміни умовних операторів оператором вибору і навпаки;

таблиці істинності логічних операцій і правилам їх запису в умовних виразах;

видам контролю при введенні даних;

Завдання 3.1. Розглянемо, як початковий приклад, програму знаходження коренів квадратного рівняння, блок-схема алгоритму якого була розглянута на рис. 1.5.

```
#include "stdafx.h"
#include <locale.h>
#include <stdio.h>
#include <math.h>

int main()
{  // Установка виводу на екран символів кирилиці
  setlocale(0,"RUS");
  float A,B,C;
  double X1,X2,D;

  printf("****************************************\n");
  printf("*     Програма знаходження коренів     *\n");
  printf("*          квадратного рівняння        *\n");
  printf("****************************************\n\n");

  printf("Введіть значення коефіцієнтів рівняння \n");
  printf("A="); scanf("%f",&A);     // введення довжини a
```

```
  printf("B="); scanf("%f",&B);     // введення довжини b
  printf("C="); scanf("%f",&C);    // введення довжини c

  if (A==0){  // !!! див. коментар до програми!!!
      // Рівняння не квадратне
      printf("Введене рівняння не є квадратним \n");
        printf("Корінь рівняння один і дорівнює %10.3f\n",-C/B);
  } else {
      // Рівняння квадратне
      D=pow(B,2)-4*A*C;
        if (D<0){
              // Дійсних коренів немає
             printf("Введене рівняння не має дійсних коренів \n");
         } else {
              // Визначаємо корені рівняння
              X1=(-B+sqrt(D))/(2*A);
              X2=(-B-sqrt(D))/(2*A);
              printf("Корені рівняння дорівнюють: X1=%7.3lf
X2=%7.3lf\n",X1,X2);
         }
  }
}
```

У даній програмі є один важливий момент, на який варто звернути особливу увагу. Це оператор **if (A==0).** На перший погляд нічого незвичайного у ньому не немає. Проте якщо пригадати, що змінна A була оголошена як дійсне число, а значить її значення зберігається в пам'яті ЕОМ з деякою точністю (в даному випадку для змінної типу float це 6 – 7 значущих цифр), то виконання операції точного порівняння для дійсних чисел найчастіше приводить до помилкових результатів.

Приклад у програмі є свого роду виключенням з правил, оскільки дійсний нуль зберігається саме як нуль, на відміну від інших значень.

Завдання 3.2. Розгляньте наступну програму і спробуйте обґрунтувати результати її роботи, які наведені після її тексту.

```
#include "stdafx.h"
#include <stdio.h>
int main()
{
	float  a=0.1;
	double b=0.1;
	if (a==0.1){
		printf("A equal 0.1\n");
	} else {
		printf("A not equal 0.1\n");
	}
	if (a==b){
		printf("A equal B\n");
	} else {
		printf("A not equal B\n");
	}
	return 0;
}
```

Виконуйте порівняння дійсних чисел завжди з урахуванням точності їх подання в машинному вигляді, наприклад так:
if (fabs(a-b)< eps), де eps — якась маленька величина, що дорівнює, наприклад, 10^{-12} для типу double.

Завдання 3.3. Написати програму, яка визначає, площа якої з трьох заданих фігур більша – прямокутника, кола або трапеції. Розміри фігур задаються такими величинами: прямокутник — довжинами сторін, коло — радіусом, трапеція — двома основами та висотою.

```
#include "stdafx.h"
#include <iostream>
#include <locale.h>

#define _USE_MATH_DEFINES
#include <math.h>

using namespace std;
int main()
{   double a,b;       // довжини сторін прямокутника
    double r;         // радіус кола
    double c,d,h;     // основи трапеції (c,d) і її висота - h
    double SP,SK,ST;  // площі прямокутника, круга, трапеції
    double SMAX;      // максимальна площа фігури
    int figure;       // ознака фігури, площа якої більша
                      // 1-прямоугольника, 2-круга, 3-трапециі

    setlocale(0,"RUS");

    cout <<"**************************************************\n";
    cout <<" Програма визначення фігури з максимальною площею \n";
    cout <<"**************************************************\n\n";
    cout <<" Введіть сторони прямокутника:\n";
    cout <<"Сторона a=";   cin>>a;
    cout <<"Сторона b=";   cin>>b;
    cout <<"Введіть радіус круга \n";
    cout <<"Радіус r=";   cin>>r;
    cout <<"Введіть параметри трапеції:\n";
    cout <<"Основа c="; cin>>c;
    cout <<"Основа d="; cin>>d;
    cout <<"Висота h="; cin>>h;

    SP=a*b;           // Визначаємо площу прямокутника
    SK=M_PI*r*r;  // Визначаємо площу круга
    ST=h*(c+d)/2;    // Визначаємо площу трапеції
```

```
    // Виводимо проміжні результати
  cout<<"\n\nПлощі дорівнюють:\n";
  cout<<" Прямокутника   = "<<SP<<"\n";
  cout<<" Круга              = "<<SK<<"\n";
  cout<<" Трапеції           = "<<ST<<"\n";

      // Вважаємо, що максимальна площа у прямокутника
      SMAX=SP; figure=1;

    // Порівнюємо поточну максимальну площу з площею круга
      if (SK>SMAX){
        // Виявилось, що площа круга більша
         SMAX=SK; figure=2;
      }

    // Порівнюємо поточну максимальну площу з площею трапеції
      if (ST>SMAX){
        // Виявилось, що площа трапеції більша
         SMAX=ST; figure=3;
      }

      // Виводимо результати на екран
      cout<<"\n Максимальна площа дорівнює "<< SMAX <<" у ";
      switch(figure) {
            case 1: cout<<" прямокутника \n";
                    break;
            case 2: cout<<" круга \n";
                    break;
            case 3: cout<<" трапеції \n";
                    break;
            default:;
      }  // case (figure)
      return 0;
}
```

4. Розробка циклічних програм

Теоретичні питання, що вимагають опрацьовування перед написанням циклічних програм, такі:

1. Поняття циклу і обґрунтування ситуацій, коли безумовно необхідно або бажано їх застосовувати.

2. Способи задавання циклів у програмах на мові С++. Правила перетворення циклів одного типу в інший.

Завдання 4.1. Як перший приклад розглянемо реалізацію алгоритму розрахунку значень функції f(x) на заданому інтервалі [a,b] з кроком dx, блок-схема якого була наведена на рис. 1.6.

Функція була визначена таким аналітичним виразом:

$$f(x)=\begin{cases}(x+1)/(x+2), & \text{при } x<0\\ (x+0.5)/x, & \text{при } x>=0\end{cases}$$

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>;
#include <locale.h>

int main()
{   double A,B,dx;
    double x;
    double y;
    double eps=1.0E-12;
    setlocale(0,"RUS");
    printf("*****************************************\n");
    printf("  Програма обчислення значень функції f(x)    \n");
    printf("   на заданому інтервалі [a,b] з кроком dx        \n");
    printf("*****************************************\n");
    printf("Введіть через пробіл значення інтервалу [a,b]:");
    scanf("%lf %lf",&A,&B);
    printf("Введіть значення кроку dx=");       scanf("%lf",&dx);
```

```
  // Цикл по значеннях X
  for (x=A; x<=B; x+=dx) {
   // Перевірка на ОДЗ
      if ((fabs(x+2)<eps)||(fabs(x)<eps)){
            printf("x=%10.6lf  Функція невизначена \n",x);
            continue;
      } // if
      y=(x<0)? ((x+1)/(x+2)):((x+0.5)/x);    // визначення значення функції
      printf("x=%10.6lf  y=%10.6lf\n",x,y); // виведення результату
  } // for (x)
  return 0;
}
```

Результати роботи програми представлені на рис. 4.1.

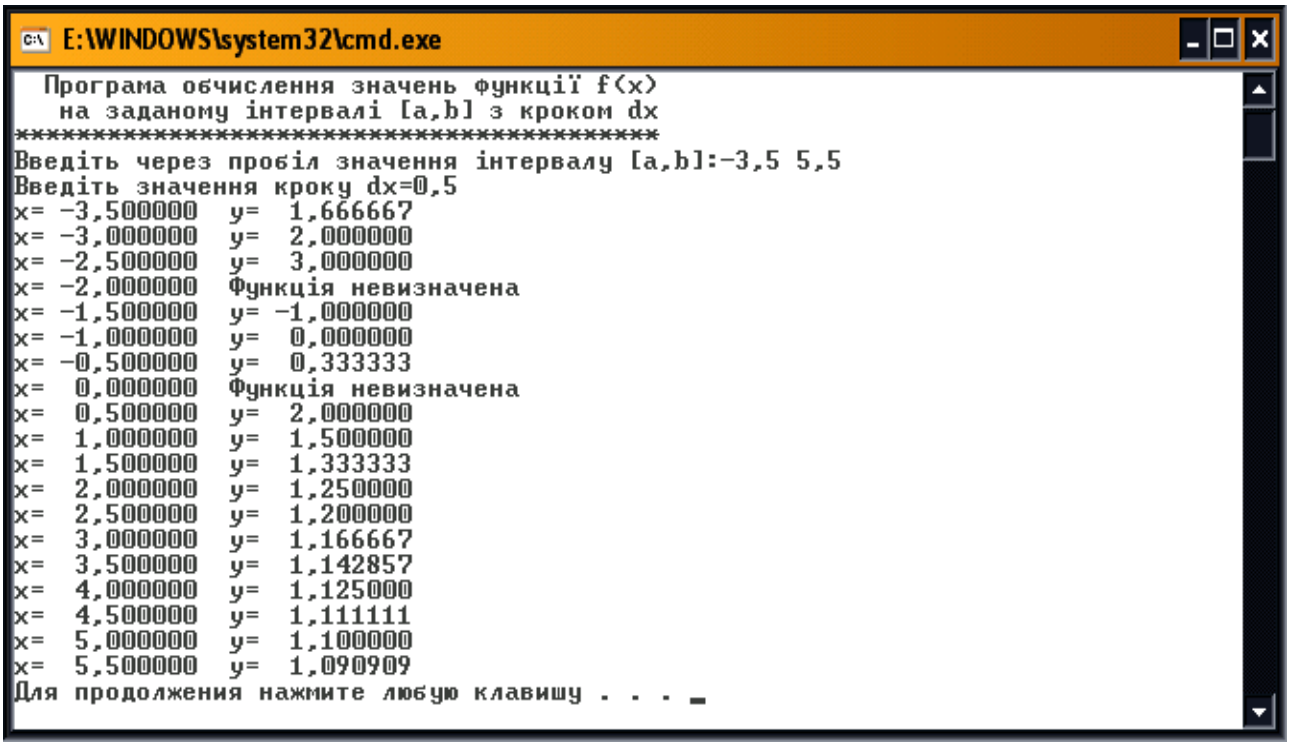


Рис. 4.1. **Результати роботи програми розрахунку значень функції на інтервалі**

Завдання 4.2. Вивести на екран таблицю множення для цілих чисел у діапазоні від N*M до (N+10)*(M+10). Значення N і M ввести з клавіатури.

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <locale.h>
#include <iomanip>

using namespace std;
int main()
{
 int N,M;
 setlocale(0,"RUS");
 cout<<"*****************************************************\n";
 cout<<"***    Програма побудови таблиці множення           ***\n";
 cout<<"***          від N*M до (N+10)*(M+10)               ***\n";
 cout<<"*****************************************************\n\n";

 // Введення початкових даних
 cout<<" Введіть значення N "; cin>>N;
 cout<<" Введіть значення M "; cin>>M;
 // Перевірка, чи займає максимальний результат більше 6 позицій
 if ((N+10)*(M+10)>999999){
        cout<<" Таблиця множення не поміститься на екрані!\n";
        return 1;
 }

 // Вивід на екран "шапки" таблиці множення
 cout<<"\n    | ";
 for(int j=M; j<M+10; j++) cout<<setw(7)<<j;
 cout<<"\n";
 for(int j=0; j<80; j++) cout<<"-";

 // Вивід на екран власне таблиці множення
 for (int i=N; i<N+10; i++) {
        cout<<setw(4)<<i<<" | ";     // виведення першої колонки
        for(int j=M; j<M+10; j++) {
                cout<<setw(7)<<i*j;    // вивід в рядок значень добутку
        } // for (j)
```

```
        cout<<"\n";            // перехід на новий рядок, при зміні (i)
 } // for (i)

 return 0;
}
```

Завдання 4.3. Побудувати таблицю основних тригонометричних функцій для кутів від 0 до 90^{o} з кроком 5 градусів.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#define _USE_MATH_DEFINES
#include <math.h>

int main()
{   double u_grad=0;     // Поточний кут в градусах
    double u_rad;        // Поточний кут в радіанах
    double du=5;         // Приріст кута в градусах

        setlocale(0,"RUS");
        // Вывод "шапки" таблицы тригонометрических функций
        printf("=========================================\n");
        printf(" Кут  |  Синус    |  Косинус  |  Тангенс          \n");
        printf("=========================================\n");
        // Розрахунок в циклі значень функцій
        while (u_grad<=90) {
              u_rad=(u_grad*M_PI)/180.;  // перетворення кута в радіани
             printf("%4.1f | %7.5f | %7.5f | %10.5f\n",
                        u_grad,sin(u_rad),cos(u_rad),tan(u_rad));
              u_grad+=du;                    // збільшуємо кут
        } // while (u_grad)
        printf("=============================================\n");
        return 0;
}
```

Виконайте дану програму. Поясніть результат, отриманий для кута в 90 градусів.

Завдання 4.4. Згенерувати таку послідовність випадкових чисел у діапазоні від -10 до 10, для якої виконується умова: сума всіх членів послідовності дорівнює сумі першого і останнього її елементів. Кількість елементів повинна бути більше 2.

```
#include "stdafx.h"
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int main()
{
   int RANGE_MIN =-10;  // мінімальне значення діапазону випад. чисел
   int RANGE_MAX = 10; // максимальне значення діапазону випад. чисел
   double rnd;         // поточне згенероване число(0..RAND_MAX)
   int rnd10;          // поточне згенероване число (-10..10)
   int i=0;            // номер згенерованого числа
   double sum_all=0,   // сума всіх чисел послідовності
                  rnd_1;     // перший член послідовності
   setlocale(0,"RUS");
   // Ініціалізація генератора випадкових чисел
   srand((unsigned)time( NULL ));
   cout<<" Послідовність випадкових чисел: \n";
   // цикл генерації випадкових чисел
   do {
        rnd =(double)rand();  // поточне згенероване число (0..RAND_MAX)
     // поточне згенероване число ( -10 .. 10)
     rnd10 = ((rnd/RAND_MAX)-0.5)*(RANGE_MAX - RANGE_MIN);
     if (i==0) rnd_1 = rnd10; // запам'ятовуємо перше число
        sum_all += rnd10;        // підраховуємо суму членів послідовності
       printf( " %4d", rnd10);  // виведення числа на екран
        i++;                              // збільшуємо номер поточного елементу
   } while ((sum_all!=(rnd_1+rnd10))||(i<3));  // перевірка умови для циклу
   printf("\nСгенеровано i=%d членів випадкової послідовності \n",i);
   return 0;
}
```

5. Розробка циклічних і розгалуджених програм

Перед написанням програм, в яких тісно переплітаються цикли й умовні оператори, необхідно повторити теоретичні питання до 3-го і 4-го практичних занять, а також вивчити способи переходу до нового кроку циклу і передчасного виходу їх циклу.

Завдання 5.1. Для заданого значення року і місяця ввести значення середніх температур за кожен день і визначити максимальну, мінімальну і середню температуру за вказаний місяць. Передбачити контроль вхідних даних на неприпустимість введення значень року і місяця більших, ніж поточні.

```
#include "stdafx.h"
#include <iostream>
#include <math.h>;
#include <locale.h>
#include <time.h>

using namespace std;
int main()
{ int year,month;                 // Рік і місяць, які вводять з клавіатури
  int cur_year, cur_month; // Поточний рік і місяць
  int kday;                        // Кількість днів в поточному місяці
  bool vis_year=false;          // Ознака високосного року
  float mint,maxt,avgt;       // Мінім., максим. і середня темпер. за місяць

  setlocale(0,"RUS");
 cout<<"*************************************************\n";
 cout<<" Програма розрахунку макс., мінім.і середн/температур  \n";
 cout<<"          за вказаний місяць і рік \n";
 cout<<"*************************************************\n\n";
```

```
//*****************************************************************
// Визначення поточного часу а потім року і місяця
time_t ltime;         // кількість секунд
struct tm today;   // структура часу

time( &ltime );    // Визначаємо к-ть секунд з 1 січня. 1970 р.
// Перетворимо секунди в структуру часу
_localtime64_s( &today, &ltime );
cur_year =today.tm_year+1900; // Перетворення до поточного року
cur_month=today.tm_mon+1;     // Перетворення до поточного місяця
cout<<" Поточний рік ="<<cur_year<< " місяць ="<<cur_month<<"\n";
//*****************************************************************

// Введення початкових даних
cout<<" Введіть значення року  : "; cin>>year;
cout<<" Введіть значення місяця: "; cin>>month;

// Перевірка на коректність введення значень
if  ((year>cur_year)||  ((year==cur_year)&&(month>=cur_month))){
        cout<<" Введені значення помилкові!\n";
         return 1;
}
// Визначаємо кількість днів в заданому місяці
switch(month){
        case  4:
        case  6:
        case  9:
        case 11: kday=30;
                 break;
        case  2: // Чи є рік високосним
                 if ((year%100==0)&&(year%400!=0)) vis_year=true;
                 kday=vis_year? 29: 28;
                 break;
        default: kday=31;
} // switch
```

```
 cout<<" Введіть значення температур за "<<kday<<" дней\n";;
 avgt=0;
 for (int i=1; i<=kday; i++){
   float temp;
   cout<<"Температура за "<<i<<"день="; cin>>temp;
        if (i==1){
            mint=maxt=temp;  // фіксуємо температуру для першого дня місяця
        } else {
           if(temp<mint) mint=temp;    // визначаємо поточну мінімальну
           if(temp>maxt) maxt=temp;   // визначаємо поточну максимальну
        }
        avgt+=temp;
 } // for (i)
 // Виведення результатів
 cout<<" Мінімальна температура за місяць     ="<<mint<<"\n";
 cout<<" Максимальна температура за місяць ="<<maxt<<"\n";
 cout<<" Середня температура за місяць          ="<<avgt/kday<<"\n";
return 0;
}
```

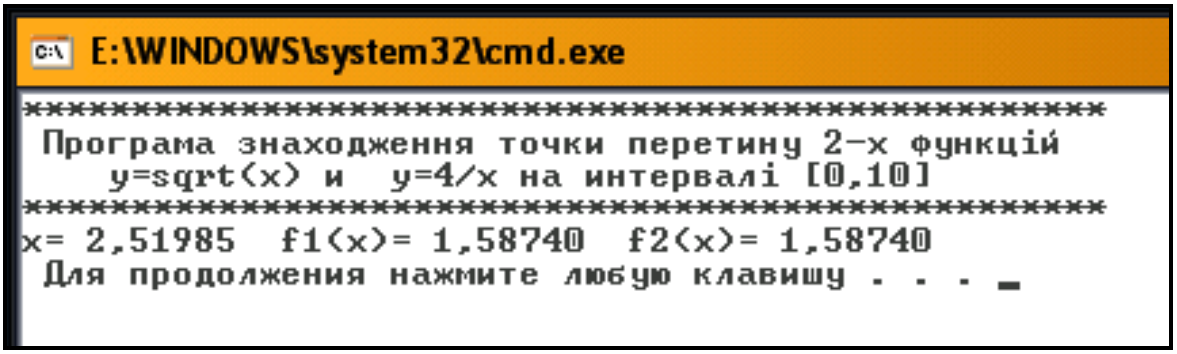
Завдання 5.2. Задано дві функції y=f1(x) і y=f2(x), що мають на інтервалі [a,b] точку перетину. Знайти значення x, в якій ці функції перетинаються.

Це завдання зводиться до класичного завдання знаходження кореня рівняння деякої функції y = F (x) на інтервалі [a,b], якщо прийняти, що F(x)= f1(x) - f2(x). Доведемо це твердження. Хай функції f1(x) і f2(x), перетинаються в точці x_0, отже $f1(x_0) = f2(x_0)$, а значить $f1(x_0) - f2(x_0) = 0$, а оскільки $F(x_0) = f1(x_0) - f2(x_0) = 0$, то x_0 — є коренем рівняння y = F(x).

Вирішимо дану задачу методом ділення інтервалу пошуку навпіл.

```
#include "stdafx.h"
#include <iostream>
#include <locale.h>
#include <math.h>
using namespace std;
int main()
{   float a=0,b=10;       // інтервал пошуку кореня рівняння
```

```
    float eps=0.00001;   // точність знаходження кореня
        float y,ya,yb;       // значення функцій в точках x, a и b.
        float x;             // значення точки  x, яке шукаємо
        setlocale(0,"RUS");
    cout<<"*************************************************\n";
    cout<<" Програма знаходження точки перетину 2-х функцій   \n";
    cout<<"    y=sqrt(x) и  y=4/x на интервалі [0,10]\n";
    cout<<"*************************************************\n";
        // Перевірка на те, що функція має різні знаки на кінцях інтервалу
    ya=sqrt(a)-(4/a);     yb=sqrt(b)-(4/b);
    if ( (ya>0)&&(yb>0) || (ya<0)&&(yb<0) ) {
        printf ("Функція має однаковий знак на кінцях інтервалу \n");
       return 1;
    }
        // Цикл пошуку кореня
        do {
            // обчислюємо значення функцій на кінцях інтервалу
            ya=sqrt(a)-(4/a);  yb=sqrt(b)-(4/b);
            x=(a+b)/2;           // визначаємо середню точку x
            y=sqrt(x)-(4/x);   // значення функції у точці x
            if ((ya<0)&&(y<0) || (ya>0)&&(y>0) ){  // знак функції у точці  x
                 a=x;
          } else {
                 b=x;
          }
        } while(fabs(a-b)>=eps);    // умова пошуку значення x з точністю eps
        // виведення результатів пошуку на екран
       printf("x=%8.5f  f1(x)=%8.5f  f2(x)=%8.5f \n ",x,sqrt(x),4/x);
        return 0;
}
```



6. Використання функцій

Для успішного написання програм з використанням функцій необхідно глибоко опрацювати теоретичний матеріал і чітко представляти відповіді на такі запитання:

1. Що таке прототип функції, для чого він потрібний?
2. Чим відрізняється оголошення функції від визначення функції?
3. Коли доцільно використовувати статичну змінну усередині функції?
4. Способи передачі параметрів у функцію.
5. Способи повернення результату з функції.
6. Поняття перевантаження функції.
7. Завдання параметрів функції за замовчуванням.
8. Правила створення заголовних файлів, що створюються користувачем.

Завдання 6.1. Створити програму, яка за допомогою функції обчислює суму трьох чисел, введених з клавіатури. Значення, що вводяться, можуть бути цілими або дійсними. Забезпечити перевантаження функцій і різний спосіб передачі параметрів.

Роботу програми організувати у вигляді простого ієрархічного меню.

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <locale.h>
#include <iostream>

using namespace std;
// Прототипи функцій  ////////////////////////////////////
char main_menu();      // Головне меню
void error_input();    // Повідомлення про помилку
char double_menu();    // Меню по дійсним функціям
void vvod_integer();   // Введення цілих чисел
```

```
void vvod_double();   // Введення дійсних чисел

// Функції обчислення суми
// цілі - за значенням
int    summa_z(int ad,     int bd,     int cd);
// дійсні - за значенням
double summa_z(double ad,  double  bd, double  cd);
// дійсні - через покажчик
double summa  (double *ad, double *bd, double *cd);
// дійсні - через посилання
double summa  (double &ad, double &bd, double &cd);
////////////////////////////////////////////////////////

// Опис глобальних змінних ( вхідні значення)
double ad,bd,cd;   // дійсні
int    ai,bi,ci;         // цілі

int main()
{   char ch;           // символ натиснутої клавіші в меню
    bool cyc=true;  // ознака виконання основного циклу

    setlocale(0,"RUS");
    while (cyc){
 ch=main_menu();   // виведення основного меню
 switch(ch) {
   case '1' : ch=double_menu(); // виведення меню дійсних чисел
                switch (ch){
                  case '1' : vvod_double();   // введення дійсних чисел
                                printf("Сума=%10.7f \n", summa_z(ad,bd,cd));
                             system("pause");
                              break;
                  case '2' : vvod_double();
                      printf("Сума=%10.7f \n", summa(&ad,&bd,&cd));
                              system("pause");
                              break;
                     case '3' : vvod_double();
                          printf("Сума=%10.7f \n",summa(ad,bd,cd));
```

```
                        system("pause");
                        break;
                    default  :;
             } // switch (2)
             break;
 case '2' : vvod_integer();  // введення цілих чисел
             printf("Сумма=%d \n",summa_z(ai,bi,ci));
             system("pause");
             break;
case 27  : cyc=false;       // закінчення роботи програми
             break;
default  : error_input();
      } // switch (1)
    } // while
   return 0;
}

// Виведення головного меню програми
char main_menu()
{
   system("CLS");
printf("**********************************\n");
printf(" 1  - Обчислення суми дійсних чисел \n");
printf(" 2  - Обчислення суми цілих чисел    \n");
printf(" Esc- Закінчення роботи програми    \n");
printf("**********************************\n");
return getch();
}

// Виведення повідомлення про помилки
void error_input()
{
  printf("Помилка при виборі варіанту!\n");
  printf("Для продовження натисніть будь-яку клавішу \n");
  getch();
}
// Вывод меню о вызове вещественных функций
```

```
char double_menu()
{
   system("CLS");
printf("*****************************************\n");
printf("1  - Функція з параметрами (за значенням)      \n");
printf("2  - Функція з параметрами (через покажчик) \n");
printf("3  - Функція з параметрами (через посиланню)\n");
printf("Esc- Відмова від обчислень \n");
printf("*****************************************\n");
return getch();
}

// Введення цілих чисел
void vvod_integer()
{
cout<<"Введіть A="; cin >> ai;
cout<<"Введіть B="; cin >> bi;
cout<<"Введіть C="; cin >> ci;
}
// Введення дійсних чисел
void vvod_double()
{
cout<<"Введіть A="; cin >> ad;
cout<<"Введіть B="; cin >> bd;
cout<<"Введіть C="; cin >> cd;
}
// Обчислення суми цілих чисел
int summa_z(int x, int y, int z)
{
return x+y+z;
}

//  Обчислення суми дійсних чисел ( за значенням)
double summa_z(double a, double b, double c)
{
return a+b+c;
}
```

```
//   Обчислення суми дійсних чисел ( через покажчик)
double summa(double *a, double *b, double *c)
{
return *a+*b+*c;
}
//   Обчислення суми дійсних чисел (через посилання)
double summa(double &a, double &b, double &c)
{
   return a+b+c;
}
```

Завдання 6.2. Написати власну функцію визначення синуса будь-якого кута із заданою точністю, використовуючи у прототипі функції параметр "за замовчуванням". Перевірити обчислення в циклі для кутів від 0 до 360 градусів, використовуючи стандартну бібліотечну функцію.

```
#include "stdafx.h"
#include <iostream>
#include <locale.h>
#define _USE_MATH_DEFINES
#include <math.h>

double mysin(double x, double eps=0.00001);

int main()
{       int ug;          // Кут в градусах
        double ur;      // Кут в радіанах
        int i,step=15;  // Крок зміни кута

        setlocale(0,"RUS");
        printf("==============================\n");
        printf("   Програма обчислення синуса кута \n");
        printf("    розкладанням в ряд Тейлора        \n\n");
        printf("    за допомогою власної функції       \n");
        printf("==============================\n");
```

```
        // Цикл обчислення і виводу на екран значень функції
        for ( ug=0; ug<=360; ug+=step){
                ur=ug*M_PI/180;
                printf("x=%4d  msin(x)=%8.6f   sin(x)=%8.6f \n",
                        ug,mysin(ur),sin(ur));
        } // for (ug)
        return 0;
}

// Власна функції обчислення синуса (x) по ряду Тейлора
// із заданою точністю eps
double mysin(double x, double eps)
{
  double msin,  // накопичувана сума ряду
         delta;     // черговий член ряду
  int n=1;         // номер кроку в циклі

  msin=delta=x;
  do {
     delta=(-1)*delta*pow(x,2)/(2*n*(2*n+1));
     msin+=delta;
     n++;
  } while(fabs(delta)>eps);   //do-while

  return msin;
}
```

Завдання 6.3. Розглянемо вже знайоме нам завдання знаходження коренів квадратного рівняння (див. завдання 3.1), проте знаходження коренів покладемо на спеціальну функцію.

Слід зазначити, що оскільки таке рівняння може мати два дійсні корені або зовсім не мати рішення, тому для такої функції краще за все передбачити повернення значень коренів через параметри, а сама функція повинна повертати ознаку того, чи має дане рівняння рішення,

чи ні.

```
#include "stdafx.h"
#include <locale.h>
#include <stdio.h>
#include <math.h>

// Прототип функції рішення квадратного рівняння
// Функція повертає число коріння ( 0- немає рішень)
// x1, x2 - знайдені корені рівняння
int square_eq(double a, double b, double c, double &x1, double &x2);

int main()
{  // Установка виводу на екран символів кирилиці
  setlocale(0,"RUS");

  float A,B,C;      // коефіцієнти рівняння
  double X1,X2;  // корені рівняння
  int rez;       // ознака результату

  printf("*************************************\n");
  printf("*    Програма знаходження коренів     *\n");
  printf("*       квадратного рівняння          *\n");
  printf("*************************************\n\n");

  printf("Введіть значення коефіцієнтів рівняння \n");
  printf("A="); scanf("%f",&A);    // введення довжини a
  printf("B="); scanf("%f",&B);    // введення довжини b
  printf("C="); scanf("%f",&C);    // введення довжини c

  // Виклик функції вирішення квадратного рівняння
  rez=square_eq(A,B,C,X1,X2);
  switch(rez){
  case 1: printf("Введене рівняння не є квадратним \n");
        printf("Корінь рівняння один і дорівнює %10.3f\n",X1);
        break;
  case 2:  printf("Корені рівняння дорівнюють: X1=%7.3lf  X2=%7.3lf
\n",X1,X2);
```

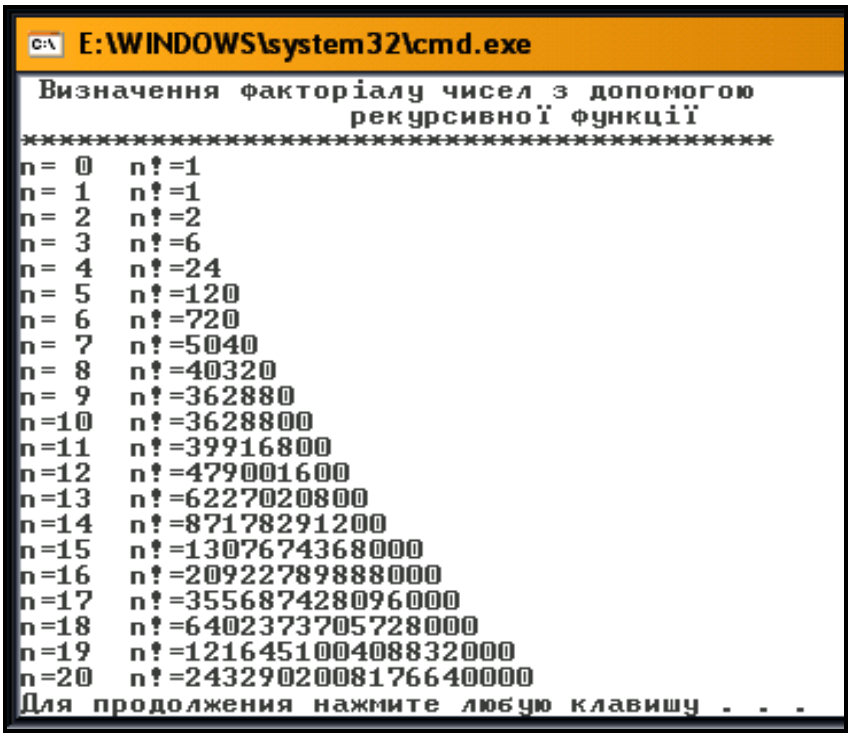
```
            break;
   default:printf("Рівняння не має рішень!\n");;
   } // switch

}

// Функція вирішення квадратного рівняння
int square_eq(double a, double b, double c, double &x1, double &x2)
{ double D;
  if (a==0){  // !!! див. коментар до програми 3.1 !!!
     // Рівняння не квадратне
  x2=x1=-c/b;  return 1;
   } else {
     // Рівняння квадратне
     D=pow(b,2)-4*a*c;
     if (D<0){
 // Дійсних коренів немає
         return 0;
     } else {
  // Визначаємо корені рівняння
  x1=(-b+sqrt(D))/(2*a);
  x2=(-b-sqrt(D))/(2*a);
  return 2;
    } // if (D)
  } // if (a)- else
}
```

Особливий інтерес слід звернути на, так звані, рекурсивні функції, які всередині свого тіла викликають саму себе. Розглянемо два приклади таких функцій.

Завдання 6.4. Визначити значення факторіала для чисел від 0 до 10. З курсу математики пригадаємо, що n! = n*(n-1)! і 0! = 1.

```
#include "stdafx.h"
#include <iostream>
```

```
#include <locale.h>
using namespace std;
// Функція обчислення факторіалу
long long fact(int n)
{   if (n<0) return -1;
else if (n<2) return 1;
else return n*fact(n-1);
}
int main()
{       setlocale(0,"RUS");
cout<<"****************************************\n";
cout<<" Визначення факторіалу чисел з допомогою \n";
cout<<"                рекурсивної функції \n";
cout<<"****************************************\n";
// Обчислення факторіалу для чисел від 0 до 20
for (int i=0; i<=20; i++) {
        printf("n=%2d  n!=%lld\n",i, fact(i));
} // for(i)
return 0;
}
```



Завдання 6.5. Скласти програму рекурсивної функції обчислення кількості сполучень $C(n,m)$, де n і m — цілі, і $0 \le m \le n$, і спираючись на неї отримати біноміальні коефіцієнти для будь-якого n.

Відомо, що

$$C(n,m)=\frac{n!}{m!\cdot(n-m)!}=\frac{n}{m}\cdot C(n-1,m-1)$$

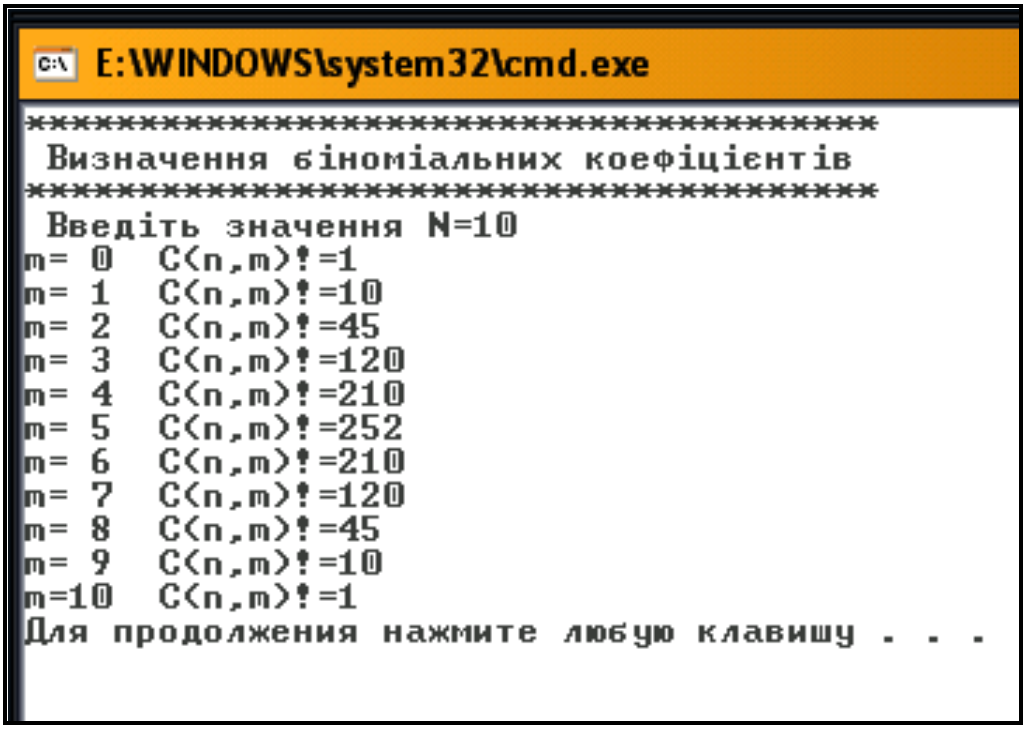
Звідси і витікає справедливість наступного рекурсивного визначення $C(n,m)$:

$$C(n,m)=\begin{cases}1, & якщо \quad m=0\\ \dfrac{n\cdot C(n-1,m-1)}{m}, & в \quad інших \quad випадках\end{cases}$$

Слід звернути увагу на те, що тут ми маємо рекурсію відразу за двома аргументами.

```
#include "stdafx.h"
#include <iostream>
#include <locale.h>

using namespace std;

// Функція обчислення числа сполучень з N по M
long long sochetan(int N, int M)

{
    if (M==0) return 1;
   else return (N*sochetan(N-1,M-1))/M;
}
```

```
int main()
{
   int N;  // значення N, що вводиться
setlocale(0,"RUS");
cout<<"*****************************************\n";
cout<<" Визначення біноміальних коефіцієнтів \n";
cout<<"*****************************************\n";


cout << " Введіть значення N="; cin>>N;
// Обчислення біноміальних коефіцієнтів
for (int i=0; i<=N; i++)
    {
       printf("m=%2d  C(n,m)!=%lld\n",i, sochetan(N,i));
} // for(i)
return 0;
}
```



7. Обробка одновимірних масивів

Масиви є одними з основних структур даних, що використовуються при вирішенні великої кількості завдань різного економічного і науково-технічного характеру. Глибоке розуміння суті масивів нерозривно пов'язане з вивченням:

способів завдання масивів на мові С++;

способів ініціалізації елементів масиву;

поняття індексу масиву й адресації елементів масиву;

способів динамічного виділення пам'яті під масив;

доступу до елементів масиву через індекс та покажчик.

Завдання 7.1. В одновимірному дійсному масиві знайти позицію (індекс) елементу, у якого найменше відхилення від середнього значення елементів масиву. Рішення задачі виконати за допомогою функцій.

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <locale.h>

// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////
// Визначення середнього значення масиву
double average(double A[], int N);
// Знаходження індексу елементу з мінім. відхиленням
int    near_index(double *A, int N, double avg);
// Виведення елементів масиву
void   vyvod (double A[],int N, int dec);
// Відображення інформації про програму
void   about_prog();

//////////////////////////////////////////////////////////
int main()
{
   // Оголошення і ініціалізація масиву
```

```
   double MAS[10]={1.0,2.3,1.12,3.4,1.1,5.3,7.0,2.31,1.11,5.12};
   double avg_mas;    // значення середнього елементів масиву

   setlocale(0,"RUS");
   avg_mas=average(MAS,10);  // визначення середнього значення
   about_prog();             // виведення інформації про програму
   vyvod(MAS,10,3);          // виведення значень масиву

   // Вывод результата программы
   printf("Середнє значення дорівнює = %10.6f\n",avg_mas);
   printf("Мінім. відхилення у елементу з індексом =%d\n",
            near_index(MAS,10,avg_mas) );
   return 0;
}


// Визначення середнього значення масиву
double average(double A[], int N)
{ double sum=0;
   for (int i=0; i<N; i++) sum+=A[i];
   return sum/N;
}

// Знаходження індексу елементу масиву A з мінім. відхиленням
// від середнього, рівного avg
int near_index(double *A, int N, double avg)
{ int ind=0;
  double delta=fabs(*(A+0)-avg);
  for ( int i=1; i<N; i++){
  if (fabs(*(A+i)-avg) < delta){
              ind=i; delta=fabs(A[i]-avg);
  } // if
  } // for (i)
  return ind;
}
```

```
// Виведення елементів масиву A з точністю в dec десяткових знаків
void   vyvod (double A[],int N, int dec)
{
printf("Масив: ");
for (int i=0; i<N; i++){
        printf(" %.*f",dec,A[i]);
}
   printf("\n");
}

// Відображення інформації про програму
void about_prog()
{
 printf("*********************************************\n");
 printf("Програма знаходження індексу елементу масиву \n");
 printf("     з найменьшим відхиленням від середнього \n");
 printf("*********************************************\n\n");
}
```

Завдання 7.2. Перетворити одновимірний масив так, щоб кожен елемент масиву дорівнював середньому поряд розташованих елементів. Вирішити задачу двома способами: з допоміжним масивом і без допоміжного масиву.

Варіант 1 (з допоміжним масивом).

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////////
// Виведення елементів масиву A з точністю в dec десяткових знаків
void   vyvod (char *name, double A[],int N, int dec);
// Відображення інформації про програму
void about_prog();
```

```
// Ініціалізація масиву випадковими числами від 0 до 100
void init_array(double *p, int n);
// Перетворення масиву
void preobr_array(double *M1,double *M2,int N);

//////////////////////////////////////////////////////////////

int main()
{   int N;                  // Розмірність масиву
    double *M1, *M2;  // Покажчики на дійсні числа (масиви)

    setlocale(0,"RUS");
    about_prog();         // інформація про програму
   // Введення розмірності масиву
   printf("Введіть розмірність масиву: "); scanf("%ld",&N);
   M1= new double[N];  // виділення динамічній пам'яті під масиви
   M2= new double[N];

   init_array(M1,N);             // ініціалізація масиву
   preobr_array(M1,M2,N); // перетворення масиву
   vyvod ("M1",M1,N,2);      // виведення початкового масиву
   vyvod ("M2",M2,N,2);      // виведення перетвореного масиву
   return 0;
}

// Виведення елементів масиву A з точністю в dec десяткових знаків
// name - ім'я масиву
void   vyvod (char *name, double A[],int N, int dec)
{
        printf("Масив %s :", name);
        for (int i=0; i<N; i++){
                printf(" %*.*f",dec+3,dec,A[i]);
        }
    printf("\n");
}
```

```
    // Відображення інформації про програму
    void about_prog()
    {
       printf("*************************************************\n");
       printf("    Програма перетворення елементів масиву  рівних    \n");
       printf(" середньому поряд розташованих елементів  (варіант №1)\n");
       printf("*************************************************\n\n");
     }

// Ініціалізація масиву випадковими числами від 0 до 100
void init_array(double *p, int n)
{
        // ініціалізація датчика випадкових чисел
        srand((unsigned)time( NULL ));
        for (int i=0; i<n; i++) {
                p[i]=((double)rand()/(double) RAND_MAX)*100.0;
        }
}

// Перетворення масиву M1 в M2
void preobr_array(double *M1,double *M2,int N)
{   double avg;
        for (int i=0; i<N; i++) {
                if (i==0) avg=M1[1]/2;                  // перший елемент
                else if (i==N-1) avg=M1[N-2]/2;   // останній елемент
                else avg=(M1[i-1]+M1[i+1])/2;    // решта елементів
                M2[i]=avg;                               // запис в новий масив
        }
}
```

Варіант 2 (без допоміжного масиву).

```
        #include "stdafx.h"
        #include <stdio.h>
        #include <math.h>
        #include <stdlib.h>
```

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

// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////////
// Виведення елементів масиву A з точністю в dec десяткових знаків
void   vyvod (char *name, double A[],int N, int dec);
// Відображення інформації про програму
void about_prog();
// Ініціалізація масиву випадковими числами від 0 до 100
void init_array(double *p, int n);
// Перетворення масиву
void preobr_array(double *M1,int N);

//////////////////////////////////////////////////////////////

int main()
{   int N;          // Розмірність масиву
    double *M1; // Покажчик на дійсне число (масив)

    setlocale(0,"RUS");
    about_prog();   // інформація про програму
    // Введення розмірності масиву
    printf("Введіть розмірність масиву: "); scanf("%ld",&N);
    M1= new double[N];  // виділення динамічної пам'яті під масив

    init_array(M1,N);                   // ініціалізація масиву
    vyvod ("исходный",M1,N,2);   // виведення перетвореного масиву
    preobr_array(M1,N);                // перетворення масиву
    vyvod ("новый   ",M1,N,2);   // виведення перетвореного масиву
    return 0;
}
// Виведення елементів масиву A з точністю в dec десяткових знаків
// name - ім'я масиву
void   vyvod (char *name, double A[],int N, int dec)
{
        printf("Массив %s :", name);
        for (int i=0; i<N; i++){
```

```
                printf(" %*.*f",dec+3,dec,A[i]);
            }
        printf("\n");
    }

// Відображення інформації про програму
 void about_prog()
 {
    printf("*****************************************************\n");
    printf("     Програма перетворення елементів масиву рівних \n");
    printf(" середньому поряд розташованих елементів(варіант №2)\n");
    printf("*****************************************************\n\n");
 }

    // Ініціалізація масиву випадковими числами від 0 до 100
    void init_array(double *p, int n)
    {
            // ініціалізація датчика випадкових чисел
            srand((unsigned)time( NULL ));
            for (int i=0; i<n; i++) {
                    p[i]=((double)rand()/(double) RAND_MAX)*100.0;
            }
    }

    // Перетворення масиву А без допоміжного
    void preobr_array(double *A,int N)
    {   double tmp;      // для зберігання перетворюваного елементу
        double pred=0;  // значення попереднього елементу масиву

      // перетворимо всі елементи, окрім останнього
      for (int i=0; i<N-1; i++) {
          tmp=A[i];               // запам'ятовуємо перетворюваний елемент
          A[i]=(pred+A[i+1])/2;  // нове значення елементу
          pred=tmp;              // перетворюваний елемент став попереднім
       } // for
      A[N-1]= pred/2;             // останній елемент масиву
    }
```

Завдання 7.3. У цілому одновимірному масиві знайти позиції двох найменших елементів і визначити суму елементів масиву, що знаходяться між ними.

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////////
// Виведення елементів масиву A
void   vyvod (int A[],int N);
// Відображення інформації про програму
void about_prog();
// Ініціалізація масиву випадковими числами від min  до max
void init_array(int *p, int n, int min, int max);
// Визначення позицій двух наїмен. елементів і суми між ними
int     reshenie(int *p,int N,int &poz1, int &poz2, int &sum);
//////////////////////////////////////////////////////////////


int main()
{   int N;      // Розмірність масиву
    int *p;     // Покажчик на ціле число (масив)
    int poz1,   // позиція 1-го мінімального елементу
    poz2,       // позиція 2-го мінімального елементу
    sum;        // сума елементів між poz1 і poz2

    setlocale(0,"RUS");
    about_prog();    // інформація про програму

        // Введення розмірності масиву
        printf("Введіть розмірність масиву: "); scanf("%ld",&N);
        p = new int[N];    // виділення динамічної пам'яті під масив
```

```
        init_array(p,N,-20,20);      // ініціалізація масиву
        vyvod (p,N);                 // виведення масиву
        if (reshenie(p,N,poz1, poz2, sum)){
            // Повідомлення про малий розмір масиву
            printf("Розмірність масиву менше 3-х елементів \n");
        } else {
       // Виведення результатів завдання
         printf("Індекси мінімальних елементів дорівнюють: %d и
%d\n", poz1,poz2);
        printf("Сума між цими елементами дорівнює: %d\n",sum);
        }
        return 0;
}

// Виведення елементів масиву A
void  vyvod (int A[], int N)
{
        printf("Масив : ");
        for (int i=0; i<N; i++){
              printf(" %3d",A[i]);
        }
   printf("\n");
}

// Відображення інформації про програму
void about_prog()
{
printf("*****************************************************\n");
 printf("Програма пошуку двох найменших елементів масиву \n");
 printf("    і визначення суми елементів між ними \n");
printf("*****************************************************\n\n");
}

// Ініціалізація масиву випадковими числами від  min  до max
void init_array(int *p, int n, int min, int max)
{
        // ініціалізація датчика випадкових чисел
```

```
        srand((unsigned)time( NULL ));
        for (int i=0; i<n; i++) {
            p[i]=(int)(((1.0*rand()/RAND_MAX)-0.5) * (max-min));
        }
}
// Визначення позицій двох наїмен. елементів (poz1, poz2)
// і суми між ними (sum)
// Повернення 1, якщо число елементів в масиві менше 3, інакше - 0
int     reshenie(int *p,int N,int &poz1, int &poz2, int &sum)
{   int tmp;            // тимчасова змінна
    int i;              // індекс циклу
    int start, finish;  // початковий і кінцевий індекс масиву для
                        // визначення суми
    if (N<3) return 1;  // повернення при малому розмірі масиву

   int min1=p[poz1=0];    // найменший елемент
   int min2=p[poz2=1];    // другий найменший елемент
   if (min1>min2){        // обмін, якщо 1-й більше другого
       min1=p[poz1=1]; min2=p[poz2=0];
   }
       // обробка решти елементів масиву
       for (i=2; i<N; i++){
           if (p[i]<min1){       // перевірка на 1-й найменший елемент
               min2=min1; poz2=poz1;
               min1=p[poz1=i];
           } else if (p[i]<min2){ // перевірка на 2-й найменший елемент
               min2=p[poz2=i];
           } // if-else
       } // for (i)
   start =((poz1<poz2)?poz1:poz2)+1;  // початковий індекс для суми
   finish=((poz1<poz2)?poz2:poz1)-1;  // кінцевий індекс для суми
   // цикл визначення суми
   for (i=start, sum=0; i<=finish; i++){
       sum+=p[i];
   } // for (i)
   return 0;
}
```

Окремо вимагають розгляду питання сортування (впорядкування) елементів масиву в певному порядку, наприклад за збільшенням. Це пов'язано з тим, що дана процедура дуже поширена і зустрічається досить часто при вирішенні конкретних завдань.

Розглянемо дві програми, в яких будуть створені функції сортування елементів масиву. У першій програмі — сортування методом "бульбашки". Метод неефективний, але тривіальний за своєю суттю і зрозумілий з погляду алгоритму реалізації. У другій програмі — швидкий алгоритм рекурсивного сортування.

Завдання 7.4. Написати програму сортування масиву методом "бульбашки".

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////////
// Виведення елементів масиву A
void   vyvod (double A[],int N);
// Відображення інформації про програму
void about_prog();
// Ініціалізація масиву випадковими числами від min  до max
void init_array(double *p, int n, int min, int max);
// Сортування масиву з N елементів методом "бульбашки"
void SortBubble(double *A, int N);
//////////////////////////////////////////////////////////////

int main()
{   int N;         // Розмірність масиву
    double *p;   // Покажчик на дійсне число (масив)
```

```
        setlocale(0,"RUS");
         about_prog();  // інформація про програму

         // Введення розмірності масиву
         printf("Введіть розмірність масиву: "); scanf("%ld",&N);
         p = new double[N];        // виділення динамічної пам'яті під масив

         init_array(p,N,0,100);      // ініціалізація масиву
         vyvod (p,N);                // виведення масиву
         SortBubble(p,N);            // сортування масиву
         vyvod (p,N);                // виведення масиву після сортування
         return 0;
}


// Виведення елементів масиву A
void   vyvod (double A[],int N)
{
         printf("Масив : ");
         for (int i=0; i<N; i++){
                  printf(" %4.1f",A[i]);
         }
    printf("\n");
}

// Відображення інформації про програму
void about_prog()
{
  printf("*****************************************************\n");
  printf("  Програма сортування масиву методом \"бульбашки\"\n");
  printf("*****************************************************\n\n");
}

// Ініціалізація масиву випадковими числами від min  до max
void init_array(double *p, int n, int min, int max)
{
         // ініціалізація датчика випадкових чисел
```

```
        srand((unsigned)time( NULL ));
        for (int i=0; i<n; i++) {
                p[i]=(int)((1.0*rand()/RAND_MAX) * (max-min+1));
        }
}

// Сортування масиву з N елементів методом "бульбашки"
void SortBubble(double *A, int N)
{   double tmp;

        for (int i=0; i<N-1; i++) {         // вибір [i]-елемента
                for (int j=i; j<N; j++) {  // порівняння з рештою елементів
                        if (A[i]>A[j]){
                                tmp = A[i];        // Обмін елементів
                                A[i]= A[j];
                                A[j]= tmp;
                        }
                } // for (j)
        } // for (i)
}
```

Завдання 7.5. Написати програму сортування масиву з використанням рекурсивного алгоритму.

Сама програма залишається незмінною, змінюється лише текст самої функції сортування, яке виглядатиме таким чином:

```
// Власне функція швидкого рекурсивного сортування
// массива для елементів від (a) до (b)

void SortRecursion(double* Mas, int a, int b)
{
  int A = a;       // межі інтервалу
  int B = b;
  double mid;   // середній елемент
  if ( b > a)    {
     // Знаходимо розділовий елемент в середині масиву
```

```
      mid = Mas[(a+b)/2];
      // Обходимо масив
      while(A<=B) {
        // Знаходимо елемент, який більше або рівний
        // розділовому елементу від лівого індексу.
         while((A<b)&&(Mas[A]<mid)) ++A;

        // Знаходимо елемент, який менше або рівний
        // розділовому елементу від правого індексу.
          while((B>a)&&(Mas[B]>mid)) --B;

        // Якщо індекси не перетинаються, міняємо елементи
        if(A<=B) {
            double T;
           T = Mas[A];
            Mas[A] = Mas[B];
            Mas[B] = T;
            ++A;   --B;
        } // if
      } // while

      // Якщо правий індекс не досяг лівої межі масиву,
      // потрібно повторити сортування лівої частини.
      if(a<B) SortRecursion( Mas, a, B );

      // Якщо лівий індекс не досяг правої межі масиву,
      //  потрібно повторити сортування правої частини.
      if(A<b) SortRecursion( Mas, A, b );
    }
  }

// Сортування масиву з N елементів швидким рекурсивним алгоритмом
void SortArray(double *A, int N)
{
   // відсортувати елементи
       SortRecursion(A, 0, N-1);
}
```

Завдання 7.6. В одновимірному масиві з N чисел знайти три такі елементи, щоб різниця між сумою цих елементів і сумою наступних за ними була максимальна.

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

#define  N   10    // розмірність масиву

// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////////
// Виведення елементів масиву A
void   vyvod (int A[],int n);
// Відображення інформації про програму
void about_prog();
// Функція пошуку 3-х елементів і різниці
int  find_3_elements (int *MAS,int n,int &i1,int & i2,int &i3, int*diff);

////////////////////////////////////////////////////////////////

int main()
{   // Оголошення і ініціалізація масиву
        int MAS[N]={2,0,5,-1,7,2,4,4,1,3};
        int i1,i2,i3;       // індекси елементів, які шукаємо
        int diff;         // максимальна різниця, яку шукаємо
        int cod=0;

        setlocale(0,"RUS");
        about_prog();         // інформація про програму
        vyvod(MAS,N);       // виведення початкового масива
        // Пошук 3-х елементів
        if ((cod=find_3_elements(MAS,N,i1,i2,i3,&diff))==-1) {
                printf("Помилка! Масив містить менш 4-х елементів \n");
```

```
        } else {
            printf("Індекси елементів дорівнюють: %d  %d  %d
\n",i1,i2,i3);
            printf("Максимальна різниця  = %d \n",diff);
        }
        return cod;
    }

    // Обчислення різниці для заданих 3-х елементів
    int DiffSum(int *A, int i1, int i2, int i3)
    {
        return (A[i1]+A[i2]+A[i3])-(A[i1+1]+A[i2+1]+A[i3+1]);
    }

    // Функція пошуку 3-х елементів і різниці
    int   find_3_elements(int *A,int n,int &i1,int & i2,int &i3, int*diff)
    {
        int cur_diff;            // поточне значення різниці

        if (n<4) return -1;      // повернення з кодом помилки
        i1=0; i2=1; i3=2;        // початкова ініціалізація індексів
        *diff=DiffSum(A,i1,i2,i3);  // початкова різниця

        for (int i=0; i<n-3; i++) {          // можливі індекси 1 елементу
          for (int j=i+1; j<n-2; j++) {      // другого елементу
            for (int k=j+1; k<n-1; k++) {  // третього елементу
                    if (*diff < (cur_diff=DiffSum(A,i,j,k))) {
                        *diff = cur_diff;      // знайдена нова трійка
                         i1=i; i2=j; i3=k;
                    } // if
            } // for (k)
          } // for (j)
        } // for (i)
        return 0;
    }
```

```
// Виведення елементів масиву A
void   vyvod (int A[],int n)
{
        printf("Масив : ");
        for (int i=0; i<n; i++) {printf(" %2d",A[i]);}
         printf("\n");
}

// Відображення інформації про програму
void about_prog()
{
  printf("*****************************************************\n");
  printf("   Програма пошуку трьох таких елементів масиву,\n");
  printf("      щоб різниця між сумою цих елементів \n");
  printf("      і сумою наступних за ними була максимальна \n");
  printf("*****************************************************\n\n");
}
```

Завдання 7.7. Виконати циклічний зсув елементів одновимірного масиву з N елементів на K позицій. Розглянемо два варіанти рішення такої задачі:

1. Реалізація циклічного зсуву масиву на 1 позицію і повторенням цієї дії K разів (неефективний алгоритм складності K*n).

2. Реалізація алгоритму завдання з використанням спеціальної функції повороту масиву [3].

Варіант 1.

```
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

#define  N   10    // розмірність масиву
```

```
// ПРОТОТИПИ ФУНКЦІЙ ////////////////////////////////////////
// Виведення елементів масиву А
void   vyvod (int A[],int n);
// Відображення інформації про програму
void about_prog();
// Зсув масиву на К елементів
void shift(int *MAS,int n,int K);

//////////////////////////////////////////////////////////////
int main()
{   // Оголошення і ініціалізація масиву
   int MAS[N]={0,1,2,3,4,5,6,7,8,9};
   int K=15;

   setlocale(0,"RUS");

        about_prog();        // інформація про програму
        vyvod(MAS,N);       // виведення початкового масива
        shift(MAS,N,K);     // зсув на К елементів
        vyvod(MAS,N);       // виведення отриманого масиву
        return 0;
}
// Зсув масиву на К элементів
void shift(int *A,int n,int K)
{   int tmp0; // тимчасові змінні
   int tmp1;
   int p;       // позиція замінюваного елементу
        for (int i=0; i<K; i++){
                tmp0=A[0];        // перший елемент запам'ятовуємо
                for (int j=0; j<n; j++) {
                    p=(j+1)%n;      // куди зсуваємо попередній елемент
                    tmp1=A[p];      // виконання зсуву
                    A[p]=tmp0;
                    tmp0=tmp1;
                } // for (j)
        } // for (i)
}
```

```
// Виведення елементів масиву A
void   vyvod (int A[],int n)
{
        printf("Масив : ");
        for (int i=0; i<n; i++){printf(" %2d",A[i]);}
        printf("\n");
}

// Відображення інформації про програму
void about_prog()
{

   printf("*************************************************\n");
   printf("   Програма циклічного зсуву елементів масиву,\n");
   printf("              на K позицій\n");
   printf("*************************************************\n\n");

}
```

Другий алгоритм, був розглянутий у книзі Джона Бентлі "Перлини програмування" [3]. Його ідея полягає в такому. Циклічний зсув масиву X зводиться фактично до заміни **ab** на **ba**, де **a —** перші **i** елементів X, а **b —** елементи, що залишилися. Завдання здається складним. Проте припустимо, що для перетворення масиву **ab** в **ba** у нас є функція **reverse**, яка переставляє елементи деякої частини масиву у протилежному порядку. У початковому стані масив має вид **ab**. Викликавши цю функцію для першої частини, отримаємо $\mathbf{a^r b}$ ($\mathbf{a^r}$ **—** це модифікована частина a, до якої застосували функцію перестановки reverse). Потім викличемо її для другої частини: отримаємо $\mathbf{a^r b^r}$. Потім викличемо функцію для всього масиву, що дасть $\mathbf{(a^r b^r)^r}$, а це в точності відповідає **ba**. Розглянемо, як буде така функція діяти на масив 0123456789, який потрібно зсунути вліво на i = 4 елементи:

```
reverse(0, i-1)   // 3210 | 456789
reverse(i, n-1)   // 3210 | 987654
reverse(0, n-1)   // 4567890123
```

Якщо зсув виконується управо, i = N - i, де N — розмірність масиву. У нашому прикладі, i = 6:

```
reverse(0, i-1)   // 543210 | 6789
reverse(i, n-1)   // 543210 | 9876
reverse(0, n-1)   // 6789012345
```

У цьому випадку частина програми, що відповідає за зсув масиву, буде такою:

```
// Функция поворота части массива
// с индексами от p0 до  p1
void reverse (int A[], int p0, int p1)
{   int tmp;
   for (int i=p0; i <=(p0+p1)/2 ; i++ ) {
        tmp=A[i];
        A[i]=A[p0+p1-i];
        A[p0+p1-i]=tmp;
  }
}

// Сдвигаємо масив на K элементів вправо
void shift(int A[N],int n,int K)
{
   K%=n; K=n-K;
   reverse(A, 0, K-1);
   reverse(A, K, n-1);
   reverse(A, 0, n-1);
}
```

8. Обробка двовимірних масивів

Обробка масивів вищого порядку вимагає часом більшої алгоритмічної "прозорливості" програміста, проте, незалежно від розміру масиву, принципи і підходи до обробки по суті ті ж, що і для одновимірних масивів. Основна принципова відмінність полягає, мабуть, у тому, що обробка масиву, який має N вимірювань базується на N вкладених циклах. Проте дуже часто обробку N-вимірних масивів можна звести до одновимірних, тим більше, що зберігаються елементи будь-якого багатовимірного масиву в послідовних елементах пам'яті. Необхідно тільки правильно записати формулу перетворення індексів (координат) багатовимірного масиву в його лінійний еквівалент і навпаки.

Завдання 8.1. Є двовимірний масив розмірності N. Побудувати в масиві трикутник Паскаля.

Нагадаємо, що елементами трикутника Паскаля є біноміальні коефіцієнти, і кожен елемент з індексом [i,j], дорівнює сумі елементів, розташованих у попередньому рядку над поточним елементів і лівіше за нього. Іншими словами A[i,j] = A[i-1,j-1]+A[i-1,j].

```
#include "stdafx.h"
#include <locale.h>
#include <iostream>
#include <iomanip>

#define      N     12   // Розмірність масиву

using namespace std;

int main()
{   int A[N][N];  // Оголошення масиву
```

```
   setlocale(0,"RUS");
        cout<<" Програма побудови трикутника Паскаля \n\n";

        // Перший стовпець заповнюємо одиницями
        for (int i=0; i<N; i++) A[i][0]=1;
        // Решту елементів заповнюємо нулями
        for (int i=0; i<N; i++)
                for (int j=1; j<N; j++ )
                        A[i][j]=0;
      // Обчислюємо значення біноміальних коефіцієнтів
        //  ( будуємо трикутник Паскаля)
        for (int i=1; i<N; i++)
                for (int j=1; j<=i; j++ )
                        A[i][j]=A[i-1][j-1]+ A[i-1][j];

   // Виводимо всі ненульові елементи на екран
        for (int i=0; i<N; i++){
                for (int j=0; j<N; j++ ){
                        if (A[i][j])   cout<<setw(6)<<A[i][j];
                } // for (j)
                cout<<"\n";
        } // for (i)
        return 0;
}
```

Завдання 8.2. Заданий прямокутний масив розмірності NxM. Кожен елемент масиву характеризує "вартість проходження" через нього, наприклад, час. Потрібно пройти від точки з координатами [0,0] до точки [N-1, M-1], відповідно до наступного правила. Перехід можна здійснювати по горизонталі вправо або по вертикалі вниз в елемент з найменшою вартістю. Визначити сумарну вартість проходження всього маршруту і видати на екран сам маршрут.

```
#include "stdafx.h"
#include <locale.h>
#include <iostream>
```

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

#define       N      6    // Розмірність масиву

using namespace std;

// Функція вибору наступної точки маршруту
int NextPoint(int A[N][N], int &i, int &j)
{
        if ((i==N-1)&&(j==N-1)) return 0; // досягли кінця
        if (i==N-1)j++;                          // у останньому рядку
          else if (j==N-1)i++;                   // у останньому стовпці
            else {
                (A[i+1][j]<A[i][j+1]? i++: j++);  // куди йти?
            }
   return 1;
}

int main()
{      int A[N][N];  // Оголошення масиву
        int randu;      // стартове число "псевдогенератора"випад.чисел
        int x,y;          // координати проходження маршруту
        int sum=0;      // сумарна вартість

       setlocale(0,"RUS");
        cout<<" Програма пошуку маршруту по вартісній матриці \n\n";

        randu=time(NULL);
        // Заповнюємо елементи "вартістю" ( випадковим чином)
        for (int i=0; i<N; i++)
                for (int j=0; j<N; j++ ){
                      A[i][j]=abs((randu) % 11+1);
                    randu*=(randu+101);
                 }

     // Виводимо масив на екран
```

```
    for (int i=0; i<N; i++){
          for (int j=0; j<N; j++ ){
               cout<<setw(4)<< A[i][j];
          }
          cout<<"\n";
    }

       x=y=0;
       cout<<" Початок маршруту = крапка (0,0)\n";

       // Пошук шляху в циклі
       while (NextPoint(A,y,x)){
          sum+=A[y][x];
          cout<<"Точка ("<<y<<","<<x<<")  Сума="<<sum<<"\n";
       }
       return 0;
}
```

Завдання 8.3. Знайти сідлову точку в матриці.

Сідловою точкою в матриці називається елемент, який є максимальним у рядку і мінімальним у стовпці. Ця точка має особливе значення в теорії ігор.

```
#include "stdafx.h"
#include <locale.h>
#include <iostream>
#include <iomanip>
#include <time.h>

#define  N   5    // Розмірність масиву N x M
#define  M  6

using namespace std;


int main()
{   int A[N][M];     // Оголошення масиву
```

```
int max, imax;  // максимальний елемент і його позиція в рядку
int jmin;       // значення стовпця сідлової точки
bool find=true;  // ознака знайденого елементу

setlocale(0,"RUS");
    cout<<" Програма пошуку сідлової точки \n\n";

    srand(time(NULL));
    // Заповнюємо елементи матриці
    for (int i=0; i<N; i++)
        for (int j=0; j<M; j++ ){
            A[i][j]=rand() %10;
        }

    // Виводимо масив на екран
    for (int i=0; i<N; i++){
      for (int j=0; j<M; j++ ){
          cout<<setw(4)<< A[i][j];
     }
      cout<<"\n";
    }
// Шукаємо максимальний елемент в кожному рядку
        for (int i=0; i<N; i++ ) {
            for ( int j=1, max=A[i][jmin=0]; j<M; j++) {
                if (A[i][j]>max){
                    max=A[i][jmin=j];
                }
            } // for (j)

            // Перевіряємо, чи є знайдений елемент
            // мінімальним у стовпці
            find=true;
            for (int im=0 ; im<N; im++ ){
                if (A[i][jmin]>A[im][jmin]) {
                    find=false; break;
                }
            } // for(im)
```

```
                    if (find) {
                    cout<<"Сідлова точка має координати ("<<i << "," 
<<jmin<<")\n";
                    break;
                    }
            } // for (i)
            if (!find)     cout<<"Сідлової точки нема!\n";
      return 0;
}
```

Завдання 8.4. Написати програму, яка виконує операцію перемножування двох прямокутних матриць. Масиви під матриці виділити динамічно за допомогою функції. Операцію перемножування матриць також реалізувати функцією.

З курсу вищої математики відомо, що перемножувати можна такі матриці, для яких виконується умова, що кількість рядків в одній матриці дорівнює кількості стовпців в іншій, і значення елементу [i,j] результуючої матриці дорівнює сумі добутків елементів і-рядка першої матриці на j-стовбець другої.

```
#include "stdafx.h"
#include <locale.h>
#include <iostream>
#include <iomanip>
#include <time.h>


using namespace std;

// Виділити динамічну пам'ять під двовимірний масив
int **Get2xArrayMemory(int n, int m)
{  static int **p;       // зберігає значення між викликами функції

  p = new int *[n];   // ** - покажчик на покажчик
  for(int i=0;i<n;i++) p[i]=new int[m];
  return p;
```

```
}

// Добуток матриць C=AxB;
void MatrixMultiplay(int **A,int **B,int **C, int N,int M,int K)
{   int sum;
        for (int i=0; i<N; i++) {     // прохід по рядках результуючої матриці
              for (int k=0; k<K; k++) { // прохід по стовпцях результуючої матриці
                    sum=0;
                    for (int j=0; j<M; j++){ // сума (рядок х стовпець)
                          sum+=A[i][j]*B[j][k];
                    } // for (k)
                    C[i][k]=sum;         // елемент результуючої матриці
              } // for (j)
        } // for (i)
}
int main()
{   int N,M,K;
    int **A, **B, **C;

    setlocale(0,"RUS");
        cout<<" Програма добутку матриць \n\n";
        cout <<" Число рядків    1-ої матриці  ="; cin>>N;
        cout <<" Число стовпців 1-ої матриці ="; cin>>M;
        cout <<" Число рядків    2-ої матриці  ="<<M<<"\n";
        cout <<" Число стовпців 2-ої матриці ="; cin>>K;
        A=Get2xArrayMemory(N,M);
        B=Get2xArrayMemory(M,K);
        C=Get2xArrayMemory(N,K);

        srand(time(NULL));
        // Заповнюємо елементи початкових матриць
        for (int i=0; i<N; i++)
              for (int j=0; j<M; j++ ){
                    A[i][j]=rand() %5;
                    for (int k=0; k<K; k++){
                          B[j][k]=rand()%4;
                    }
```

```
            }

        // Виводимо початкові масиви на екран
        for (int i=0; i<N; i++){        // Масив A
          for (int j=0; j<M; j++ ){
                cout<<setw(4)<< A[i][j];
         }
          cout<<"\n";
        }

   cout<<"\n";
        for (int i=0; i<M; i++){        // Масив B
          for (int j=0; j<K; j++ ){
                cout<<setw(4)<< B[i][j];
       }
         cout<<"\n";
        }
        MatrixMultiplay(A,B,C,N,M,K);

        // Виведення добутку матриць
        cout<<"Результат добутку дорівнює:\n";
        for (int i=0; i<N; i++){        // Масив C
          for (int j=0; j<K; j++ ){
                cout<<setw(4)<< C[i][j];
         }
          cout<<"\n";
        }
        return 0;
}
```

Завдання 8.5. Написати програму рішення системи лінійних рівнянь за допомогою методу Гауса – Жордана.

Алгоритм цього методу такий:

1. Вибирають першу колонку зліва, в якій є хоч одне відмінне від нуля значення.

2. Якщо найбільш верхнє число в цій колонці є нуль, то міняють весь перший рядок матриці з іншим рядком матриці, де в цій колонці немає нуля.

3. Усі елементи першого рядка ділять на верхній елемент вибраної колонки.

4. З рядків, що залишилися, віднімають перший рядок, помножений на перший елемент відповідного рядка, з метою отримати першим елементом кожного рядка (окрім першого) нуль.

5. Далі проводять таку ж процедуру з матрицею, що виходить з початкової матриці після викреслювання першого рядка і першого стовпця.

6. Після повторення цієї процедури n-1 разів отримують верхню трикутну матрицю.

7. Віднімаємо з передостаннього рядка останній рядок, помножений на відповідний коефіцієнт, з тим, щоб у передостанньому рядку залишилася тільки 1 на головній діагоналі.

8. Повторюють попередній крок для подальших рядків. У результаті отримують одиничну матрицю і рішення на місці вільного вектора (з ним необхідно проводити усі ті ж перетворення).

```
#include "stdafx.h"
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
// Введення системи рівнянь
double **Vvod_SLAU(int &N);
// Виведення системи рівнянь
void Vyvod (double**M, int N);
// Рішення СЛАУ
int  ReshenieSLAU(double **A, int N);

int main()
{   int N;          // Розмірність системи рівнянь
   double **A;  // Покажчик на динамічний двовимірний масив

       setlocale(0,"RUS");
      A=Vvod_SLAU(N);        // Введення системи рівнянь
       Vyvod(A,N);
```

```
        if (ReshenieSLAU(A,N)==0){
                // Виведення коренів рівнянь
                printf("n\Корні СЛАУ дорівнюють: \n");
                for (int i=0; i<N; i++){
                        printf("X%d = %7.4lf\n",i+1,A[i][N]);
                }
        } else {
                printf("***  Порядок системи рівнянь менший N  ***\n");
                printf("або вона має нескінченну безліч рішень!\n");
        }
        return 0;
}
// Введення системи рівнянь
double **Vvod_SLAU(int &N)
{ static double **M;
  printf("Уведіть порядок СЛАУ : ");    scanf("%ld",&N);
  // Виділення пам'яті під СЛАУ
  M = new double *[N];     // ** - покажчик на покажчик
  for(int i=0;i<N;i++) M[i]=new double[N+1];
  // Введення значень СЛАУ
  printf("Значення в рядку вводіть через пробіл!\n");
  for (int i=0; i<N; i++){
          for(int j=0; j<=N; j++) {
              scanf("%lf",&M[i][j]);
          } // for(j)
  } // for(i)
  return M;
}

// Виведення системи рівнянь
void Vyvod (double**M,int N)
{ double elm;

  printf("\n Початкове рівняння: \n");
  for (int i=0; i<N; i++){
          for(int j=0; j<N; j++) {
             printf("%+6.2lf*X%d ",M[i][j],j+1);
```

```
        } // for(j)
    printf(" = %+6.2lf\n",M[i][N]);
  } // for(i)
}
// Обмін рядків, якщо поточний елемент головної діагоналі рівний нулю
int FindNoNullLine(double**A,int N,int i){
        double tmp;
        if (A[i][i]!=0)       return 0;   // не нуль - нормальне повернення
        for (int r=i+1; r<N; r++){
                if (A[r][r]!=0){                        // найшли рядок
                        for(int j=0; j<=N; j++) {   // міняємо рядки
                                tmp=A[i][j]; A[i][j]=A[r][j]; A[r][j]=tmp;
                        }
                        return 0;
                } // if
        } // for (r)
        return 1;                // помилка
}
// Рішення СЛАУ
int  ReshenieSLAU(double **A, int N)
{   double elm;  // елемент головної діагоналі
    double kf;     // коеф. на який множимо рядок
        for (int i=0; i<N; i++) {
                if(A[i][i]==0) {
                        if (FindNoNullLine(A,N,i)) return 1;
                }
                // Ділимо рядок на елемент [i,i]
               elm=A[i][i];
                for (int j=0; j<=N; j++) A[i][j]/=elm;
                // Віднімаємо і-строку з решти рядків
                for(int i1=0; i1<N; i1++ ){
                    if (i1==i) continue;
                     kf=A[i1][i];
                    for (int j=0; j<=N; j++) A[i1][j]-=A[i][j]*kf;
                } // for(i1)
        } // for(i)
        return 0;
```

```
}
```

9. Самостійний розбір програм і аналіз алгоритмів

Уміння розробити правильний алгоритм рішення поставленої задачі і реалізувати його у вигляді працюючої програми є дуже важливим, проте не менш важливим часто стає завдання іншого плану — розібрати наявну готову програму, що реалізує певний алгоритм, можливо оптимізувати його, виправити приховані помилки і т. п.

Розглянемо ряд постановок завдань і реалізацію їх рішення [4] на мові C++. Текст програми приводиться без коментарів. Метою завдання є розбір алгоритму рішення задачі на основі аналізу тексту програми і виконання докладного коментаря у програмі.

Паралельно слід змінити структуру тексту програми так, щоб підвищити його читабельність.

Якщо в тексті програми зустрічається оператор **goto**, то слід продумати, чи є можливість реалізувати запропоновані алгоритми без використання цього оператора?

Завдання 9.1. Надрукувати всі прості числа, що не перевершують задане число M.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

#define N 150

int main()
{
// Прості до M
```

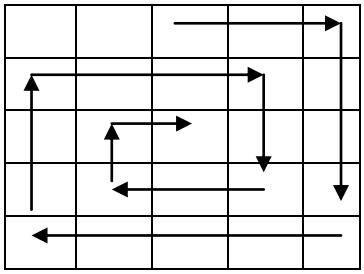
```
int m,i,j,k,q,P[N];

scanf("%d",&m);
if (m>=2) printf("2\n");
if (m>=3) printf("3\n");
P[k=0]=3;
for (i=5; i<=m; i+=2)
  { for (j=0; j<=k; j++)
    { q=P[j]; if (q*q>i) break;
    if (!(i%q)) goto NP;
    }
  printf("%d\n",i);
  if (k<N-1) P[++k]=i;
  NP:;
 }
    return 0;
}
```

Завдання 9.2. Скласти програму виведення всіх тризначних десяткових чисел, сума цифр яких дорівнює даному натуральному числу.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

// Задана сума цифр
int main()
{ int N, i,j, k;
scanf("%d",&N);
for (i=0; i<=9; i++)
for (j=0; j<=9; j++)
{ k=N-i-j;
if (k>=1 && k<=9) printf(" %d\n",i+10*j+100*k);
}
return 0;
```

```
}
```

Завдання 9.3. Ввести число N і заповнити двовимірний масив розміром NxN числами 1, 2 ... по спіралі



```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

#define NN 19
#define AK A[i][j]=++k; if (k==n*n) goto REZ;
// Спіраль
int main()
{ int i,j,k,n,A[NN][NN];
scanf("%d",&n);
k=i=j=0;
for ( ; ; )
{ do {AK; j++;} while (i+j<n-1);
do {AK; i++;} while (i<j);
  do {AK; j--;} while (i+j>n-1);
  do {AK; i--;} while (i>j+1);
}
REZ: for (i=0; i<n; i++)
{ for (j=0; j<n; j++) printf("%4d",A[i][j]);
printf("\n");
}
}
```

Завдання 9.4. Надрукувати всі чотиризначні натуральні числа, в десятковому записі яких немає двох однакових цифр.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

// Числа з різних цифр
int main()
{ int i, j, k, m;
for (i=1; i<=9; i++)
for (j=0; j<=9; j++) if (i!=j)
for (k=0; k<=9; k++) if (k!=i && k!=j)
for (m=0; m<=9; m++)
if (m !=i && m!=j && m!=k)
printf(" %d\n",(((i *10+j)*10+k)*10) +m);
}
```

Завдання 9.5. Напечатати в порядку зростання всі прості нескоротні дроби між 0 і 1, знаменники яких не перевищують 7.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

// Впорядковані дроби
int main()
{ int P, n, m, i, j, a, b;
scanf("%d",&P);
m=0; n=1;
do
```

```
{ printf("%d/%d\n",m,n);
  i=j=1;
for (b=2; b<=P; b++)
{ a=m*b/n+1;
if (a*j<b*i) {i=a; j=b;}
}
m=i; n=j;
}
while (i<j);
}
```

Завдання 9.6. Натуральне число називається досконалим, якщо воно дорівнює сумі всіх своїх власних дільників, включаючи 1. Надрукувати всі досконалі числа, менші, ніж задане M.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

// Досконалі числа
int main()
{ int m, i, j, k, s;
scanf( "%d", &m);
for (i =2; i <=m; i ++)
{ s=j=1;
do
{ j++;
k=i/j;
if (i==k*j && j<=k)
    {s+=j; if (j<k) s+=k;}
} while (j<k);
if (s==i) printf("%d\n",i);
}
}
```

Завдання 9.7. Надрукувати всі представлення натурального числа N сумою натуральних чисел. Перестановка доданків нового способу не дає.

```
#include "stdafx.h"
#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>

#define NN 100

// Розкладання на доданки

int main()
{ int N,M[NN],i,k, t,s;
  scanf(" %d",&N); printf (" %d\n", N);
  M[0]=N; k=i=0;
  RA:
  t=M[k]-1;
  s=t+i-k+1;
  for (i=k; i<N; i++)
  if (s>t) {M[i]=t; s-=t;}
  else {M[i]=s; break;}
  for (k=0; k<=i; k++)
  printf ("%d ",M[k]);
  printf("\n");
  for (k=i; k>=0; --k)
  if(M[k]>1) goto RA;
}
```

10. Контрольні завдання і завдання для самостійного вирішення

10.1. Контроль основних операцій і порядку їх виконання

Завдання 10.1. Що надрукує наведена нижче програма [12] ?

```
main()
{
  int x;

  x = - 3 + 4 * 5 - 6;   printf("%d\n",x);
  x = 3 + 4 % 5 - 6;     printf("%d\n",x);
  x = - 3 * 4 % - 6 / 5; printf("%d\n",x);
  x = ( 7 + 6 ) % 5 / 2; printf("%d\n",x);
}
```

Завдання 10.2. Що надрукує наведена нижче програма?

```
#define PRINTX printf("%d\n",x)
main()
{
   int x=2,y,z;

   x *= 3 + 2;  PRINTX;
   x *= y = z = 4; PRINTX;
   x = y == z; PRINTX;
   x == ( y = z ); PRINTX;
}
```

Завдання 10.3. Що надрукує наведена нижче програма?

```
#define PRINTX(int) printf("%d\n",int)


main()
{
   int x,y,z;
   x = 2; y = 1; z = 0;
   x = x && y || z; PRINTX(x);
   PRINTX(x || y && z);
   x = y = 1;
   z = x ++ - 1;  PRINTX(x); PRINTX(z);
   z+=-x++ + ++y; PRINTX(x); PRINTX(z);
   z=x/++x; PRINTX(z);
}
```

Завдання 10.4. Що надрукує наведена нижче програма?

```
#define PRINT(int) printf("int = %d\n",int)

main()
{
  int x,y,z;
  x = 03; y = 02; z = 01;
  PRINT( x | y & z );
  PRINT( x | y & ~ z );
  PRINT( x ^ y & ~ z );
  PRINT( x & y && z );

  x = 1; y = -1;
  PRINT( ! x | x );
  PRINT( ~ x | x );
  PRINT( x ^ x );
  x <<= 3;  PRINT(x);
  y <<= 3;  PRINT(y);
  y >>= 3;  PRINT(y);
}
```

Завдання 10.5. Що надрукує наведена нижче програма?

```
#define PRINT(int) printf("int = %d\n",int)

main()
{
   int x=1, y=1, z=1;
   x += y += z;
   PRINT( x < y ? y : x );

   PRINT( x < y ? x++ : y++ );
   PRINT( x ); PRINT( y );

   PRINT( z += x < y ? x++ : y++ );
   PRINT( y ); PRINT( z );

   x = 3; y = z = 4;
   PRINT( ( z >= y >= x )? 1 : 0 );
   PRINT( z >= y && y >=x );
}
```

Завдання 10.6. Що надрукує наведена нижче програма?

```
#define PRINT3(x,y,z)
        printf("x=%d\ty=%d\tz=%d\t\n", x, y, z)
main()
{
   int x, y, z;

   x = y = z = 1;
   ++x || ++y && ++z; PRINT3(x, y, z);

   x = y = z = 1;
   ++x && ++y || ++z; PRINT3(x, y, z);

   x = y = z = 1;
   ++x && ++y && ++z; PRINT3(x, y, z);

   x = y = z = -1;
   ++x && ++y || ++z; PRINT3(x, y, z);
```

```
    x = y = z = -1;
    ++x || ++y && ++z; PRINT3(x, y, z);
    x = y = z = -1;
    ++x && ++y && ++z; PRINT3(x, y, z);
}
```

10.2. Контроль вміння складати блок-схеми алгоритмів вирішення завдань

Завдання 10.7. Дана сторона рівностороннього трикутника. Знайти площу цього трикутника.

Завдання 10.8. Відомі гіпотенуза і катет прямокутного трикутника. Знайти другий катет і радіус вписаного кола.

Завдання 10.9. Відома довжина кола. Знайти площу круга, обмеженого цим колом.

Завдання 10.10. Знайти суму членів арифметичної прогресії a, a+d, …a+(n-1)d за даними значеннями a, d, n.

Завдання 10.11. Дані дійсні позитивні числа a, b, c. За трьома сторонами з довжинами a, b, c можна побудувати трикутник. Знайти кути трикутника.

Завдання 10.12. Визначите число, отримане виписуванням у зворотному порядку цифр заданого цілого тризначного числа x.

Завдання 10.13. Дано натуральне число n, що складається з шести цифр. Визначите число сотень і тисяч у ньому.

10.3. Контроль написання і відладки лінійних програм

Скласти алгоритм і програму на мові C++ для обчислення:

Завдання 10.14. Площі грані, площі повної поверхні і об'єму куба при заданій довжині ребра.

Завдання 10.15. Відстані між двома точками, заданими своїми координатами.

Завдання 10.16. Площі сектора, радіус якого рівний r, а відповідний центральний кут рівний f, кут заданий у радіанах.

Завдання 10.17. Площі сектора, радіус якого рівний r, відповідний центральний кут рівний f, кут заданий у градусах.

Завдання 10.18. Добуток цифр заданого чотиризначного числа.

10.4. Обчислювальні алгоритми, що розгалуджуються

Скласти алгоритми рішення і програми на C++ для таких завдань:

Завдання 10.19. Дані дійсні числа a, b, c. Подвоїти ці числа, якщо a≥b≥c, і замінити їх абсолютними значеннями, якщо це не так.

Завдання 10.20. Дані три дійсні числа. Звести в квадрат ті з них, значення яких не негативні.

Завдання 10.21. Дані дійсні позитивні числа x, y, z. З'ясувати, чи існує трикутник з довжинами сторін x, y, z. Якщо трикутник існує, то відповісти — чи є він гострокутним.

Завдання 10.22. Дані дійсні числа a, b, c. Перевірити, чи виконуються нерівності a < b < c.

10.5. Циклічні обчислювальні алгоритми

Завдання 10.23. Послідовність Фібоначчі задається законом: $a_1=1$; $a_2=1$; $a_n+1= a_n + a_n-1$, n = 3, 4, . . . Отримати перші 10 членів послідовності. Отримати члени послідовності, менші 100. Підрахувати їх суму і кількість.

Завдання 10.24. Знайти суму квадратів всіх цілих чисел від a до b (значення a і b вводяться з клавіатури; b ≥ a).

Завдання 10.25. Скласти програму зведення натурального числа у квадрат, враховуючи таку закономірність:

$1^2 = 1$,
$2^2 = 1 + 3$,
$3^2 = 1 + 3 + 5$,
$4^2 = 1 + 3 + 5 + 7$,
……

Завдання 10.26. Громадянин 1 березня відкрив рахунок у банку, вклавши 1 000 грн. Через кожен місяць розмір внеску збільшується на 2 % від наявної суми. Визначити:

а) приріст суми внеску за перший, другий ..., десятий місяць;

б) суму внеску через три, чотири ..., дванадцять місяців.

Завдання 10.27. Розглянути послідовність, утворену дробами 1/1, 2/1, 3/2 ..., у якій чисельник (знаменник) наступного члена послідовності виходить складанням чисельників (знаменників) двох попередніх членів. Чисельники двох перших дробів рівні 1 і 2, знаменники — одиницям. Знайти перший член такої послідовності, який відрізняється від попереднього не більше ніж на 0,001.

Завдання 10.28. Задане натуральне число. Визначити, чи є воно членом послідовності Фібоначчі.

Завдання 10.29. Знайти суму цілих позитивних чисел, кратних 3, менших 100 і великих 20.

10.6. Використання функцій у програмі

Скласти алгоритми і програми вирішення таких завдань з використанням функцій:

Завдання 10.30. Дані дійсні числа x, y, z. Знайти max(x,y,z), min(x,y,z) і їх суму, використовуючи алгоритм-функцію знаходження max/min двох чисел.

Завдання 10.31. Використовуючи алгоритм-функцію знаходження найбільшого загального дільника двох натуральних чисел (НЗД) знайти НЗД: а) трьох натуральних чисел; б) чотирьох натуральних чисел; в) N натуральних чисел.

Завдання 10.32. Використовуючи алгоритм-функцію знаходження найменшого загального кратного двох натуральних чисел знайти НЗК: а) трьох натуральних чисел; б) чотирьох натуральних чисел; в) N натуральних чисел.

Завдання 10.33. Є два натуральні числа. З'ясувати в якому з них: а) більше цифр; б) менша сума цифр, використовуючи алгоритм-функцію для знаходження кількості цифр (або суми цифр).

Завдання 10.34. Упорядкувати за зростанням значення трьох змінних, використовуючи процедуру SWAP-зміни місцями двох змінних.

Завдання 10.35. Існують два натуральні числа. Визначити, чи є одне число перевертишем іншого, використовуючи процедуру отримання числа, зворотного даному.

Завдання 10.36. Для двох заданих натуральних чисел визначити, чи є ці числа взаємно простими, використовуючи процедуру знаходження найбільшого загального дільника двох чисел (алгоритм Евкліда).

10.7. Обробка масивів

Завдання 10.37. Сформувати з елементів одновимірного масиву, що стоять на парних і непарних місцях, два масиви. Знайти середні арифметичні елементів цих масивів;

Завдання 10.38. З одновимірного масиву вибрати елементи, що не належать сегменту [m - k], де m, k — задані цілі числа, і сформувати з них новий масив.

Завдання 10.39. Вставити в одновимірний масив заданий елемент на задану позицію.

Завдання 10.40. Вставити заданий елемент (після) максимального в масиві. Якщо максимальних декілька, то після першого.

Завдання 10.41. У масиві зберігаються відомості про кількість опадів, що випали за кожен день серпня. Визначити: а) у який період випало більше опадів: у першу половину серпня чи в другу; б) у яку декаду місяця випало більше всього опадів.

Завдання 10.42. Дано масив ненульових цілих чисел. Визначити, скільки разів елементи масиву при перегляді від його початку змінюють знак. Наприклад, в масиві 11 -14, 22, 56 -24, -99 знак змінюється три рази.

Завдання 10.43. У масиві з 20 елементів числа утворюють неспадну послідовність. Декілька елементів, що йдуть підряд, рівні між собою. Знайти кількість таких елементів.

Завдання 10.44. У масиві з 30 елементів числа утворюють

неспадну послідовність. Знайти кількість різних чисел у масиві.

Завдання 10.45. Дано масив цілих чисел. Розглянути відрізки масиву (групи чисел, які йдуть підряд), що складаються з непарних чисел. Отримати найбільшу з довжин даних відрізків. Вирішити окремо для одновимірного і двовимірного масиву.

Завдання 10.46. Дано масив цілих чисел. Визначити кількість парних елементів і кількість елементів, що закінчуються на цифру 5. Вирішити окремо для одновимірного і двовимірного масиву.

Завдання 10.47. Дана цілочисельна прямокутна матриця. Визначити номер першого із стовпців, що містять хоч б один нульовий елемент.

Характеристикою рядка цілочисельної матриці назвемо суму її негативних парних елементів. Переставляючи рядки заданої матриці, розташувати їх відповідно до спадних характеристик.

Завдання 10.48. Упорядкувати рядки цілочисельної прямокутної матриці за збільшенням кількості однакових елементів у кожному рядку.

Знайти номер першого із стовпців, що не містять жодного негативного елементу.

Завдання 10.49. Шляхом перестановки елементів квадратної матриці із дійсних елементів добитися того, щоб її максимальний елемент знаходився в лівому верхньому кутку, наступний за величиною — в позиції (2,2), наступний — в позиції (3,3) і т. д., заповнивши таким чином всю головну діагональ.

Знайти номер першого з рядків, що не містять жодного позитивного елементу.

Рекомендована література

1. Арсак Р. Программирование игр и головоломок / Арсак Р. — М. : Мир, 1994. — 224 с.

2. Ахо А. Построение и анализ вычислительных алгоритмов / Ахо А., Хопкрофт Дж., Ульман Дж. — М. : Мир, 1979. — 536 с.

3. Бентли Дж. Жемчужины программирования / Бентли Дж. — СПб. : Питер, 2002. — 272 с.

4. Брудно А. Л. Московские олимпиады по программированию / Брудно А. Л., Каплан Л. И. ; под. ред. акад. Б. Н. Наумова. — М. : Наука, 1990. — 208 с.

5. Виленкин Н. Я. Популярная комбинаторика / Виленкин Н. Я. — М. : Наука, 1975. — 208 с.

6. Вирт Н. Алгоритмы и структуры данных / Вирт Н. ; пер. с англ. — М. : Мир, 1989. — 360 с.

7. Грегори К. Использование Visual C++ 6 / Грегори К. ; пер. с англ. — М. ; СПб. ; К. : Вильямс, 2000. — 864 с.

8. Дейтел Х. Как программировать на C++ / Дейтел Х., Дейтел П. ; пер. с англ. — М. : ЗАО "Издательство БИНОМ", 1998. — 1024 с.

9. Джамса К. Учимся программировать на языке C++ / Джамса К. ; пер. с англ. — М. : Мир, 1997. — 320 с.

10. Задачи по программированию / Абрамов С. А., Гнездилова Г. Г., Капустина Е. Н., Селюн М. И. — М. : Наука, 1988. — 224 с.

11. Зелковиц М. Принципы разработки программного обеспечения / Зелковиц М., Шоу А., Гэннон Дж. ; пер. с англ. — М. : Мир, 1982. — 386 с.

12. Керниган Б. Язык программирования Си. Задачи по языку Си / Керниган Б., Ритчи Д., Фьюер А. — М. : Финансы и статистика, 1985, — 279 с.

13. Кнут Д. Искусство программирования для ЭВМ. Т.1—3 / Кнут Д. — М. : Мир, 1976—1977.

14. Ковальски Р. Логика в решении проблем / Ковальски Р. — М. : Наука, 1990. — 280 с.

15. Кристофидес Н. Теория графов / Кристофидес Н. М. — М. : Мир, 1979. — 215 с.

16. Крячков А. В. Программирование на С и С++ / Крячков А. В., Сухинина И. В., Томшин В. К. — М. : Горячая линия-Телеком, 2000. — 344 с.

17. Липский В. Комбинаторика для программистов / Липский В. — М. : Мир, 1989. — 213 с.

18. Логика и компьютер. Моделирование рассуждений и проверка правильности программ / Алешина Н. А., Анисов А. М., Быстров П. И., Непейвода Н. Н. — М. : Наука, 1990. — 240 с.

19. Малоземов В. В. Рекурсивные вычисления / Малоземов В. В. — Ленинград : ЛГУ, 1978.

20. Марченко А. Л. С++. Бархатный путь / Марченко А. Л. — М. : Горячая линия-Телеком, 1999. — 400 с.

21. Мейерс С. Эффективное использование С++ / Мейерс С. ; пер. с англ. — М. : ДМК, 2000, — 236 с.

22. Подбельский В. В. Язык С++ : учебн. пособ. / Подбельский В. В. — М. : Финансы и статистика, 1995. — 560 с.

23. Практическое руководство по программированию / Б. Мик, П. Хит, Н. Рашби и др. ; под ред. Б. Мика, П. Хит, Н. Рашби ; пер. с англ. — М. : Радио и связь, 1986. — 168 с.

24. Разработка приложений на Microsoft Visual C++ 6. Учебный курс: Официальное пособие Microsoft для самостоятельной подготовки : пер. с англ. — М. : Издательско-торговый дом "Русская редакция", 2000. — 576 с.

25. Райзер Дж. Комбинаторика / Райзер Дж. — М. : Мир, 1969. — 154 с.

26. Рейнгольд Э. Комбинаторные алгоритмы / Рейнгольд Э., Нивергельт Ю., Део Н., — М. : Мир, 1980. — 477 с.

27. Страуструп Б. Дизайн и эволюция языка С++ / Страуструп Б. ; пер. с англ. — М. : ДМК, 2000. — 444 с.

28. Страуструп Б. Язык программирования С++ / Страуструп Б. ; пер. с англ. — 3-е изд. — СПб. ; М. : Невский Диалект — БИНОМ, 1999. — 991 с.

29. Фокс Дж. Программное обеспечение и его разработка / Фокс Дж. ; пер. с англ. — М. : Мир, 1985. — 368 с.

30. Фридман А. Л. Основы объектно-ориентированного программирования на языке С++: учебный курс / Фридман А. Л. — М. : Радио и связь, 1999. — 205 с.

31. Холл М. Введение в комбинаторику / Холл М. — М. : Мир, 1970. — 424 с.

32. Шилдт Г. Самоучитель С++ / Шилдт Г. ; пер. с англ. — СПб. : BHV-Санкт-Петербург, 1998. — 620 с.

33. Эллис М. Справочное руководство по языку С++ с комментариями / Эллис М., Страуструп Б. ; пер. с англ. — М. : Мир, 1992. — 445 с.

34. Язык компьютера / пер. с англ. под ред. и с предисл. В. М. Курочкина. — М. : Мир, 1989. — 240 с.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації
до виконання практичних завдань з навчальної дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"
для студентів напряму підготовки
"Комп'ютерні науки"
всіх форм навчання

Укладачі: **Тарасов Олександр Васильович**
Федорченко Володимир Миколайович
Лосєв Михайло Юрійович

Відповідальний за випуск **Пономаренко В. С.**

Редактор **Дуднік О. М.**

Коректор **Байдак В. В.**

План 2010 р. Поз. № 202.
Підп. до друку Формат 60 х 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 5,75. Обл.-вид. арк. 7,2. Тираж прим. Зам. №

Видавець і виготівник — видавництво ХНЕУ, 61001, м. Харків, пр. Леніна, 9а

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи ***Дк № 481 від 13.06.2001 р.***

Методичні рекомендації
до виконання практичних завдань
з навчальної дисципліни
"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

для студентів напряму підготовки
"Комп'ютерні науки"
всіх форм навчання