Міністерство освіти і науки України
Чернігівський державний технологічний університет

ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ

Методичні вказівки

до виконання лабораторних робіт для студентів напрямку підготовки 0908 ”Електроніка” по спеціальності 6.090803 “Електронні системи”

Частина 2

Мова програмування “С”

ЧЕРНІГІВ ЧДТУ 2007

Міністерство освіти і науки України
Чернігівський державний технологічний університет

ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ

Методичні вказівки

до виконання лабораторних робіт для студентів напрямку підготовки 0908 ”Електроніка” по спеціальності 6.090803 “Електронні системи”

Частина 2

Мова програмування “С”

Затверджено
на засіданні кафедри
промислової електроніки
протокол №5 від 19.01.07р.

ЧЕРНІГІВ ЧДТУ 2007

Програмування та алгоритмічні мови. Методичні вказівки до виконання лабораторних робіт для студентів напрямку підготовки 0908 ”Електроніка” по спеціальності 6.090803 “Електронні системи”. Частина 2. Мова програмування “С”/ Укл. Ревко А.С., Гордієнко В.В. –Чернігів: ЧДТУ, 2007. – 60 с.

Укладачі: Ревко Анатолій Сергійович, кандидат технічних наук, доцент кафедри
Гордієнко В’ячеслав Валентинович, кандидат технічних наук, доцент кафедри

Відповідальний за випуск: Денисов Ю. О., завідувач кафедри промислової електроніки, кандидат технічних наук, доцент

Рецензент: Бивойно П. Г., кандидат технічних наук, доцент Чернігівського державного технологічного університету

ЗМІСТ

ВСТУП

Ці методичні вказівки призначені для студентів першого курсу за напрямом підготовки “Електроніка” (0908) по спеціальності “Електронні системи” (6.090803) денної форм навчання та слугують для допомоги студентам під час підготовки та виконання лабораторних робіт з дисциплін “Програмування та алгоритмічні мови” у другому семестрі при вивченні мови програмування “С”.

Метою проведення лабораторних робіт є опанування мови програмування "С", набуття навичок в розробці алгоритмів методом покрокової деталізації на основі базових логічних структур, вивчення та освоєння основних методів та прийомів програмування, отримання практичних навичок роботи за комп’ютером по відлагодженню та тестуванню програм.

Цикл лабораторних робіт виконується протягом семестру і охоплює основні теми курсу "Програмування та алгоритмічні мови" другого семестру. Теоретичною основою для виконання лабораторних робіт є курс лекцій, теоретичні відомості на початку кожної лабораторної роботи даних методичних вказівок, практичні заняття та навчальна література.

Ефективність мови "С" в самих різноманітних прикладних програмах, від вирішення невеликих задач чисельного характеру до розробки складних програмних систем – компіляторів, баз даних, операційних систем, програмування мікроконтролерів т.ін. обумовила її широке розповсюдження та популярність. Мова відобразила найважливіші концепції технології розробки програм: розвинута система типів даних, орієнтація на принципи структурного програмування, підтримка процесу покрокової розробки. Основною метою лабораторних робіт є опанування мови програмування С, вивчення та освоєння основних методів та прийомів програмування на цій мові, отримання практичних навичок роботи за комп'ютером по відлагодженню та тестуванню програм.

Описи лабораторних робіт виконані по єдиній структурі та включають до себе мету роботи, короткі теоретичні відомості, методичні вказівки для підготування до роботи та її виконання, індивідуальні завдання до лабораторної роботи, контрольні запитання для підготування до захисту роботи. Тематика лабораторних робіт охоплює практично всі основні типи даних та основні оператори мови "С".

Для допущення до виконання лабораторної роботи студенту необхідно підготувати заздалегідь план виконання роботи, який повинен включати:

1) Назву лабораторної роботи.

2) Тему лабораторної роботи.

3) Короткі теоретичні відомості.

4) Методичні вказівки.

5) Завдання для лабораторної роботи свого варіанту з відповідними вказівками.

6) Блок-схеми програм.

Після виконання лабораторної роботи для її захисту необхідно скласти звіт, який в себе включає: пункти 1-6 плану роботи, текст програми, результати виконання вказівок та тестів по програмі, висновок по роботі.

Лабораторна робота захищається на лабораторному занятті або на консультації (за дозволом викладача). Студент, що не захистив дві попередні роботи не допускається до наступної. Також до лабораторної роботи не допускається студент, який не має плану проведення поточної лабораторної роботи.

Після виконання лабораторних робіт студент складає звіт по циклу лабораторних робіт, який складається з титульного листа, змісту та всіх звітів по кожній лабораторній роботі. Приклад титульного листа звіту приведений у додатку Б.

Під час захисту лабораторної роботи студент повинен відповісти на декілька питань, які приведені в кінці кожної роботи в методичних вказівках, а також на питання безпосередньо по тексту та блок-схемі програми.

Після успішного виконання та захисту всіх лабораторних робіт, а також здавання звіту по циклу лабораторних робіт викладачу, студент допускається до складання іспиту чи заліку.

Для підготування до виконання та захисту лабораторних робіт можна використовувати конспект лекцій, ці методичні вказівки, іншу літературу по мові програмування "С". Список рекомендованої літератури наведений в кінці методичних вказівок.

1 ЛАБОРАТОРНА РОБОТА №1. ОПЕРАТОРИ УМОВНОГО ПЕРЕХОДУ ТА ОПЕРАТОРИ ЦИКЛІВ

Мета роботи: отримання навичок з використання умовних операторів, операторів циклів, операторів вводу – виводу у програмі; отримання навичок з використання простих типів даних; програмування циклічних обчислювальних процесів.

1.1 Теоретичні відомості

Поняття ідентифікатора. Ідентифікатор використовується в якості імені об'єкта (функції, змінної, константи і т.д.). Ідентифікатори повинні вибиратися з урахуванням наступних правил:

а)обов'язково починається з букви латинського алфавіту (a,..., z. A,..., Z) або з символу підкреслювання (_), в них можуть використовуватися букви латинського алфавіту, символ підкреслювання і цифри (0,...,9). Використання інших символів в ідентифікаторах заборонено;

б)букви нижнього регістру (a, ..., z), що використовуються в ідентифікаторах, відрізняються від букв верхнього регістру (A, ..., Z). Це значить, що наступні ідентифікатори вважаються різними: prog, ProG, PROG, pRoG і т.д.;

в)ідентифікатори можуть включати будь-яку кількість символів, із яких сприймаються і використовуються для виявлення різних об'єктів (імен) тільки перші 32.

Типи даних і оголошення змінних. Програма оперує з різними даними, котрі можуть бути простими і структурованими. Прості дані – це цілі і дійсні числа, текст та вказівники (містять адреси пам'яті, за якими розміщуються дані). В мові розрізняють поняття опису змінної і її визначення (об'яву). Опис встановлює властивості об'єкта: його тип, розмір і т.д. Визначення разом з тим викликає виділення пам'яті. Кожний тип даних визначається ключовими словами, які приведені в таблиці 1.1.

Приклади оголошення даних:

int *a,b;* ***unsigned*** *i,j;* ***float*** *k;* тут оголошені цілі *a* и *b*, беззнакові цілі *i* і *j*, дійсне число одинарної точності *k*.

Математичні функції (модуль <*math.h*>):

abs*(x)* (***fabs***,***cabs***)– модуль цілого (дійсного, комплексного) аргументу;

pow*(x,y)* (***powlv***) – *x* в степенні *y* (x^y);

sqrt*(x)* (***sqrtl***) – квадратний корінь з *x*;

pow10*(x)* (***pow10l***) *10* в степені *x* (10^x);

ldexp*(x,y)* (***ldexpl***) – $x * 2^y$;

exp*(x)* (***expl***) –значення експоненціальної функції (e^x);

log*(x)* (***logl***) – значення натурального логарифма *x*;

log10*(x)* (***log10l***)) – значення логарифма з основою *10*;

sin*(x)* (***sinl***) – синус *x*;

Таблиця 1.1 – Прості типи даних

Назва типу	Тип значення змінної	Діапазон значень	Пам'ять, у бітах	Примітки
Int	Цілий	-32768 ...32767	16	Задає значення, до яких відносяться всі цілі числа, наприклад –6, 0, 28 і т.д.
short	Короткий і цілий	-32768 …32767	16	Об'єкти ***short*** не можуть бути більше, ніж ***int***. В Borland C ***int*** і ***short*** рівної довжини
long	Довгий і цілий	-2147483647 ..2147483647	32	Використовуються, коли діапазон значень виходить за границі діапазону типу ***int***
char	Символь-ний	Символи кодової таблиці ASCII (0...255)	8	Задає значення, котрі представляють різноманітні символи, наприклад, *w, y, ф, 4, !, ., ** і т.д. Цей тип часто використовують як найменше беззнакове ціле значення
unsigned	Беззнако-вий			Модифікатор типів ***char***, ***short***, ***int***, ***long***, що визначає їх беззнаковыми[1)]
float	Дійсний	±3.4e-38… ±3.4e+38	32	Визначає дійсні числа, дробова частина котрих відділяється крапкою (наприклад, -5.27, 0.0, 31.69 і т.д.). Дійсні числа можуть записуватися в експоненціальній формі. Наприклад: -1.58e+2 ($-1{,}58 * 10^2$), 3.61e-4 ($3{,}61 * 10^{-4}$).
double	Дійсний, подвійна точність	±1.7e-308... ±1.7e+308	64	Визначає дійсні змінні подвійної точності, що займають в два рази більше пам'яті, чим змінна типу ***float***

[1)]Типи ***unsigned*** (наприклад ***unsigned char***) можуть приймати більші по абсолютній величині додатні значення, ніж змінні знакових типів, за рахунок використання знакового біта для представлення числа. Наприклад, змінна типу ***unsigned int*** може приймати значення від 0 до 65535 (просто ***int*** при тому ж розмірі в бітах від –32768 до 32767). По замовчуванню ***unsigned*** *a* визначається як ***unsigned int*** *a*.

***cos**(x)(**cosl**)* – косинус *x* ;
***tan**(x) (**tanl**)* – головне значення тангенса ;
***asin**(x) (**asinl**)* – головне значення арксинуса *x*;
***acos**(x) (**acosl**)* – головне значення арккосинуса *x*;
***atan**(x) (**atanl**)* – головне значення арктангенса *x*;
***atan2**(x,y) (**atan2l**)* – головне значення арктангенса *x/y*;
***sinh**(x) (**sinhl**)* – синус гіперболічний *x;*
***cosh**(x) (**coshl**)* – косинус гіперболічний *x* ;
***tanh**(x) (**tanhl**))* – тангенс гіперболічний *x*;
***ceil**(x) (**ceill**)* – округлення до більшого значення;
***floor**(x) (**floorl**)* – округлення до меншого значення;
***fmod**(x,y) (**fmodl**)* – ціла частина від ділення двох чисел (*x/y*);
***modf**(x,i) (**modfl**)* – ціла і дробова частини *x*.

Програми на мові Сі зазвичай складаються з деякої кількості окремих функцій (підпрограм), серед яких повинна бути одна з ім'ям ***main***. З цієї функції починається виконання програми. Більш докладніше про це описано в третій лабораторній роботі.

Операторні дужки. Операторними дужками **(**складеним оператором) у Сі є фігурні дужки – *'{' '}'* (аналог ***begin end*** у Турбо Паскалі).

Введення – виведення інформації. У мові Сі є ряд функцій, що призначені для реалізацій операцій введення – виведення. Найбільше використовується функція форматованого виводу:

printf("керуючий рядок", список_змінних_через_кому);

Формат ***printf*** включає в себе як текстові повідомлення, так і символи керування. Символам керування передує символ %, за яким можуть слідувати букви, що визначають прототип виводу значень змінних. Вибір прототипу залежить від типу змінної, значення якої буде виводитись замість прототипу. Основні прототипи змінних перераховані у таблиці 1.2.

Таблиця 1.2 – Основні прототипи змінних

Назва типу	Формат	Примітки
char	***%c***	
char[n]	***%s***	(Рядок – масив символів), де *n* – кількість символів у рядку.
int	***%d***	
long	***%ld***	
float	***%f***	
double	***%lf***	

Кількість форматів у масці виведення повинно відповідати кількості змінних у списку змінних після лапок. Змінні розділяються між собою комами. До формату можуть входити також спеціальні символи: **\n** – новий рядок, **\t** – табуляція, \\ – вивід символу \, **\"** – вивід символу **"** та інші.

Символи, що не є символами формату чи спецсимволами, безпосередньо виводяться функцією ***printf***.

Приклад використання оператора ***printf*** для виводу значень змінних *a,b*:

```
int a,b;                          // оголошення змінних a,b
printf("a=%d, b=%d;\n",a,b);      // виведення значень змінних a,b
                                  //у формі a=5, b=10;
```

Оператор введення призначений для введення значень змінних з клавіатури. Частіше за все використовується форматоване введення: *scanf(“формат“,X1,…Xn);*

Формат оператора ***scanf*** відповідає формату оператора ***printf***. Відмінність полягає в тому, що перед значеннями змінних усіх типів, за виключенням масивів (рядків символів) та вказівників, ставиться амперсант – символ ***“&.”***. Він значить, що в розпорядження функції надається не вміст, а адреса змінної, куди треба записати значення.

Приклад використання оператора ***scanf*** для вводу значень змінних *a,b* цілого типу:

```
int a,b;                          // оголошення змінних a,b
scanf ("%d%d", &a, &b );          // введення значень змінних a,b з клавіатури
printf("a=%d b=%d\n", a, b);      // виведення значень змінних a,b
```

Для використання функцій ***scanf*** і ***printf*** необхідно підключити бібліотеку <*stdio.h*>.

Логічні операції. В мові Сі для роботи з логічними операторами прийняті декілька основних варіантів позначення операцій порівняння, які представлені у таблиці 1.3. У мові Сі нема окремого логічного типу даних. В якості нього використовуються значення цілого типу. Нуль відповідає значенню *False*, а будь-яке не нульове значення – *True*.

Таблиця 1.3 – Логічні операції порівняння

Познач	Операція
!=	Не дорівнює
==	Дорівнює
<	Менше
>	Більше
<=	Менше або дорівнює
>=	Більше або дорівнює
&&	Логічне І (виконується, якщо всі умови виконані)
||	Логічне АБО (виконується, якщо хоча б одна умова виконана)

Засоби розгалуження. Всі вирази, що реалізують умови, у конструкції вибору, повинні братися в круглі дужки.

Якщо необхідно перевірити декілька умов, то кожна умова береться у свої дужки, а між дужками ставляться логічні оператори (в залежності від

логіки). У випадку виконання декількох умовних операторів ці оператори беруться у фігурні дужки (операторні дужки). Зверніть увагу, що перед оператором не ставиться крапка з комою.

Оператор умовного переходу слугує для виконання деякого оператора в залежності від умови. Умовний оператор складається з ключового слова ***if,*** за яким у круглих дужках іде вираз цілого типу і потім оператор_1. За оператором може йти ключове слово ***else*** через **;** і оператор_2.

Виконання умовного оператора полягає в обчисленні виразу, якщо його значення *True* (не дорівнює нулю), то виконується оператор_1, якщо значення виразу-умови *False* (дорівнює нулю), і умовний оператор не містить слова ***else***, то його виконання завершується. Якщо є слово ***else***, то виконується оператор_2.

Загальні вигляди умовного оператора:

if (<вираз>) <оператор_1>;

if (<вираз>) <оператор_1>; ***else*** <оператор_2>;

Перша форма називається скороченою, друга – повною формою умовного оператора.

Там, де синтаксис мови вимагає використовувати оператор, може стояти і складений (блок операторів, взятих у фігурні дужки), і пустий оператор (символ «;» – крапка з комою).

Приклад:

```
#include <stdio.h>      // підключення бібліотеки stdio.h
void main()             // основна функція main
{  int a;               // оголошення змінної a
   scanf("%d",&a);      // введення значення змінної a з клавіатури
   if(a==3)             // порівняння змінної a з 3
   printf("a = 3");     // виведення повідомлення на екран у випадку
                        // виконання умови
 else printf("a не = 3"); //виведення повідомлення на екран у випадку
}                       // невиконання умови
```

Умовна операція. Умовна операція може успішно використовуватися замість конструкції ***if else*** там, де оператори, що входять до неї, є простими виразами. Синтаксис умовної операції такий:

<результат>=<вираз>***?***<вираз_1>***:***<вираз_2>***;*** Для прикладу розглянемо програму. Змінній *result* під час її ініціалізації буде присвоєно значення *b*, якщо вираз (a<0) істинний, і *a*, якщо вираз (a<0) неістинний. В прикладі значення змінної *result* залежить від введеного значення змінної *a*:

```
#include <stdio.h>              // підключення бібліотеки stdio.h
void main()                     // основна функція main
{int a,b=0;                     // оголошення змінної a
scanf("%d",&a);                 // введення значення змінної a з клавіатури
int result=(a<0)?b:a;           // оголошення змінної result за умовою
printf("a=%d b=%d result=%d\n",a,b,result);   // виведення значень
}                                             // змінних a,b, result
```

Оператор *switch*. Конструкція ***switch*** заміняє розгалужений багаторазовий оператор ***if else***. Синтаксис оператора ***switch*** такий:

switch (<вираз>) {
case <константний-вираз_1>***:***<оператор(и)>; ***break***;
case <константний-вираз _2>***:***<оператор(и)>; ***break***;
……
case <константний-вираз _n>***:***<оператор(и)>; ***break***;
default:<оператор(и)>; ***break***;
}

Після виконання виразу у заголовку оператора його результат послідовно порівнюється з константними виразами, починаючи з самого верхнього, доки не буде встановлено їх відповідність. Тоді виконуються оператори всередині відповідного ***case***, керування переходить на наступний константний вираз, і виконуються всі оператори до кінця ***switch***. Саме тому в кінці кожної послідовності операторів повинен бути присутнім оператор ***break***. Після виконання послідовності операторів всередині одної гілки ***case***, що завершується оператором ***break***, відбувається вихід із оператора ***switch***. Зазвичай оператор ***switch*** використовується тоді, коли програміст хоче, щоб була виконана тільки одна послідовність операторів із декількох можливих.

Кожна послідовність операторів може містити нуль чи більше окремих операторів. Фігурні дужки в цьому випадку не потрібні. Гілка, що називається ***default*** (замовчування), може бути відсутня. Якщо вона є, то послідовність операторів, що стоять безпосередньо за словом ***default*** і двокрапками, виконується тільки тоді, коли порівняння «виразу» ні з жодним з константних виразів, що стоять вище, (у ***case***) не істинне. Приклад:

```
#include <stdio.h>                    // підключення бібліотеки stdio.h
void main()                           // основна функція main
{int a;                               // оголошення змінної a
scanf("%d",&a);                 // введення значень змінної a з клавіатури
switch(a)
{     case 3:printf("a = 3");         // виведення повідомлення на екран
                 break;               // у випадку a=3
      case 4:printf("a = 4");         // виведення повідомлення на екран
                 break;               // у випадку a=4
      default: printf("a = %d\n",a); // виведення значення змінної a
}}
```

Цикли. Цикли, або ітераційні структури, дозволяють повторювати виконання окремих операторів чи групи операторів. Кількість повторень в деяких випадках фіксовано, а в інших визначається в процесі роботи на основі одної чи декількох перевірок умов.

Цикли завершуються в наступних випадках: 1)перетворилось у нуль умовний вираз у заголовку циклу; 2)у тілі циклу виконався оператор ***break***; 3)в тілі циклу виконався оператор ***return***.

В перших двох випадках керування передається на оператор, що розташований безпосередньо за циклом. В третьому випадку відбувається повернення із функції (завершення роботи даної функції).

Бувають цикли з перевіркою умови перед початком виконання тіла циклу (з передумовою), по закінченню виконання тіла (з післяумовою) чи всередині тіла. Розглянемо всі вказані типи циклів.

Цикл *while*. Синтаксис циклу ***while*** (доки):

while (<умовний_вираз>)<оператор>;

У циклі типу ***while*** перевірка умови відбувається перед виконанням тіла циклу (оператор). Якщо результат вирахування умови виразу не дорівнює нулю, то виконується оператор (чи група операторів). Перед входженням до циклу ***while*** в перший раз зазвичай ініціалізують одну чи декілька змінних для того, щоб умовний вираз мав якесь значення. Оператор чи група операторів, що складають тіло циклу, повинні, як правило, змінювати значення одної чи декількох змінних, що входять в умовний вираз, щоб в кінцевому випадку вираз перетворився в нуль і цикл завершився.

Потенціальною помилкою при програмуванні циклу ***while***, як і циклу будь-якого іншого типу, є запис такого умовного виразу, яке ніколи не завершить виконання циклу. Такий цикл називається безкінечним (наприклад цикл: *while(a) printf(“Circle”),* де *a* – будь-яке число, відмінне від 0. Цикл буде безкінцево виводити на екран монітора текст *Circle*).

Приклад:

```
#include <stdio.h>          // підключення бібліотеки stdio.h
void main(void)             // основна функція main
{       int a;              // оголошення змінної a
        scanf("%d",&a);     // введення значення змінної a з клавіатури
        while(a>=0)         // цикл повторюється, доки a>=0
        { printf("a=%d\n",a); // виведення значення змінної a
        a--; }              // зменшення значення змінної a на один
}
```

Цикл *do while*. У циклі ***do while*** перевірка умови виконується після виконання тіла циклу. Синтаксис циклу:

do <оператор>; ***while*** (<умовний_вираз>);

У мові Сі замість одиночного оператора може бути підставлена група операторів (блок). Цикл закінчує виконуватись, коли умовний вираз перетворюється в нуль (становиться неістинним).

Приклад:

```
#include <stdio.h>          // підключення бібліотеки stdio.h
void main(void)             // основна функція main
{ int a;                    // оголошення змінної a
        scanf ("%d",&a);    // введення значення змінної a з клавіатури
        do {                // початок циклу
             printf("a=%d\n",a);// виведення значення змінної a
             a--; }         // зменшення значення змінної a на 1
```

```
        while(a>=0);            // цикл повторюється доки a>=0
}
```

Цикл *for*. Найбільш загальною формою циклу у мові Сі є цикл ***for***. Цикл ***for*** – це більш загальна і більш потужна форма, ніж аналогічні цикли в мовах Паскаль і Бейсик. Конструкція ***for*** виглядає наступним чином:

for (<вираз 1>; <вираз 2>; <вираз 3>) <оператор>;

Кожного із трьох виразів можна не вказувати. Хоча, впринципі, кожний з цих виразів може бути використаний програмістом як завгодно, зазвичай перший вираз слугує для ініціалізації параметру, другий – для виконання перевірки на закінчення циклу, а третій вираз – для зміни значення параметру. Формально це правило можна описати так:

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

Виконання в будь якому місці тіла циклу оператора ***continue*** призводить до миттєвого переходу до кроку 4.

Приклад:

```
#include <stdio.h>              //підключення бібліотеки stdio.h
void main(void)                 //основна функція main
{       int a;                  //оголошення змінної a
        for(a=0;a<10;a++)       //цикл від 0 до 9-ти
        printf("a=%d\n",a);     //виведення значення змінної a
}
```

Цикл ***for*** можна звести до циклу ***while*** наступним чином:

<вираз 1>;
while (<вираз 2>) {
<оператор>;
<вираз 3>; }

Типові помилки в програмах:

1) Після круглих дужок операторів *for* и *while* крапка з комою не ставиться (інакше буде пусте ціло циклу).

2) Кількість відкриваючих "{" і закриваючих "}" дужок у програмі і в будь-якій функції повинне бути однаковим. Для цього під час створення програм існує правило – якщо відкривається якась дужка, вона зразу ж закривається, а весь необхідний вміст пишеться між дужками.

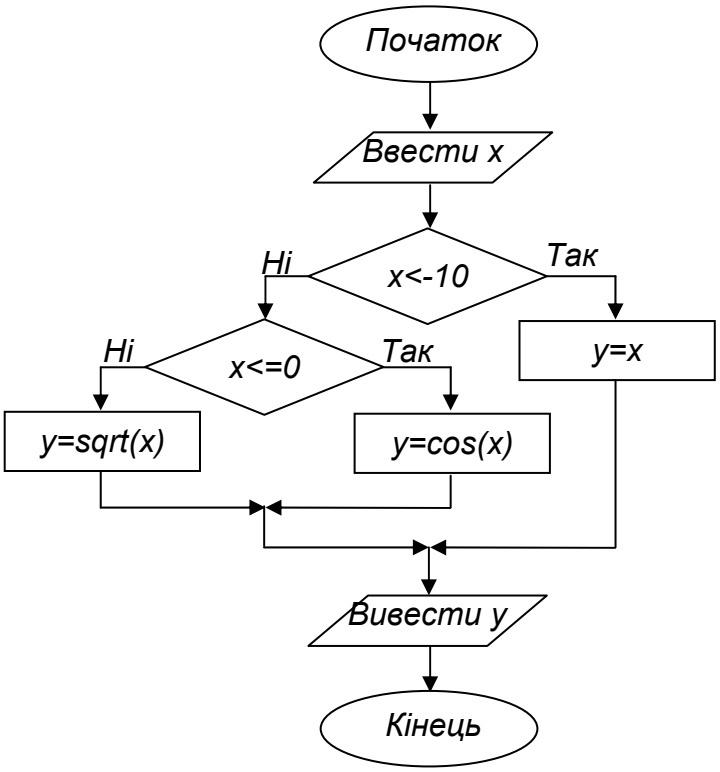
Приклади вирішення лабораторного завдання:

Задача1: Знайти значення функції *y=f(x)* за наступною формулою:

$$f(x)=\begin{cases} x, & \text{ÿêùî } x<-10; \\ \cos x, & \text{ÿêùî } -10\le x\le 0; \\ \sqrt{x}, & \text{ÿêùî } x>0. \end{cases}$$

Блок-схема і текст програми:

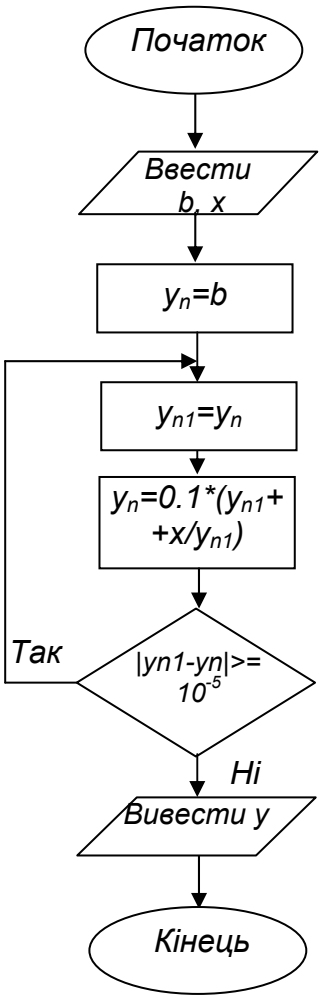
```
#include <stdio.h>
#include <math.h>
#include <conio.h>
void main()
{
 float x,y; clrscr();
 printf("Введіть значення x:");
 scanf("%f",&x);
 if (x<-10) y=x; else
  if (x<=0) y=cos(x); else y=sqrt(x);
 printf("y=%f",y);
 getch();
}
```



Задача2: Дано додатні дійсні числа b,*x*. В послідовності $y_1, y_2,\ldots, y_n$ яка утворена за законом: $y_0=b$, $y_i = 0.1(y_{i-1} + \frac{x}{y_{i-1}})$, *i=1,2,..,n*. Знайти перший член y_n, для якого виконується нерівність $|y_n - y_{n-1}| < 10^{-5}$.

Блок-схема і текст програми:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
void main()
{
 float x,b,yn,yn1;
 clrscr();
 printf("Введіть значення b і x:");
 scanf("%f%f",&b,&x);
 yn=b;
 do
   yn1=yn,yn=0.1*(yn1+x/yn1);
 while (fabs(yn-yn1)>=10e-5);
 printf("y=%f",yn1);
 getch();
}
```



1.2 Методичні вказівки:

а) вивчити основні типи даних та допустимі операції зі змінними цих типів;

б) вивчити види умовних операторів, операторів вводу – виводу, синтаксичні особливості написання та правил роботи кожного з них;

в) розробити алгоритми розв`язання задач свого варіанту, записавши їх у вигляді блок-схем;

г) програма повинна правильно виконуватися за будь-яких припустимих значеннях початкових даних;

д) правильність роботи всіх гілок програми повинна бути перевірена на тестах;

е) вивчити можливості мови програмування для реалізації обчислювальних процесів циклічної структури з відомим числом повторень у циклі та коли число повторень невідомо;

ж) розробити алгоритми розв`язання задач свого варіанта, записавши їх у вигляді блок-схем;

з) передбачити усі можливі ситуації в програмі так, щоб уникнути входження в нескінчений цикл (зациклення);

и) забезпечити в програмі виведення коментарів, відповідних різним можливим ситуаціям.

1.3 Контрольні запитання:

1. Оператори умовного переходу.
2. Синтаксис повного та скороченого оператора умовного переходу.
3. Написати фрагмент програми по приведеній блок-схемі з використанням операторів умовного переходу.
4. Прості типи, операції над змінними цих типів.
5. Оператори вводу – виводу.
6. Оператор циклу з параметром.
7. Оператор циклу з передумовою.
8. Оператор циклу з післяумовою.

1.4 Варіанти завдань

Варіант 1

Задача 1: Дано дійсні числа *a,b,c,d*. Якщо $a \le b \le c \le d$, то кожне число замінити найбільшим з них; якщо $a>b>c>d$ то числа залишити без змін; в іншому випадку всі числа замінити їх квадратами.

Вказівки:

1. Значення *a,b,c,d* задати самостійно.
2. Виконати програму для трьох випадків:
   a) $a \le b \le c \le d$; b) $a>b>c>d$; c) $a<b>c<d$.
3. На екран вивести початкові та змінені значення *a,b,c,d*.

Задача 2: Написати програму для знаходження загального опору паралельного та послідовного з'єднання *n* резисторів.

Вказівки: Значення *n* та опори резисторів вводити з клавіатури.

Варіант 2

Задача 1: Дано дійсні числа *x,y*. Якщо *x,y* від`ємні, то кожне значення замінити його модулем; якщо від`ємне лише одне число, то обидва значення збільшити на *0.5*; якщо обидва значення невід`ємні і жодне з них не належить до відрізка [*0.5; 2.0*], то обидва значення зменшити у *10* разів; в інших випадках *x,y* залишити без змін.

Вказівки:

1. Значення *x,y* задавати самостійно.
2. Виконати програму для випадків:
   a) $x<0$ і $y<0$; b) $x<0, y>0$; або $x>0, y<0$; c) $0<x<0.5; y>0$;
   d) $0.5 \le x \le 2$ і $0.5 \le y \le 2$.
3. На екран вивести початкові дані *x,y* та змінені.

Задача 2: Метод послідовних наближень дозволяє вирахувати, для знаходження кореня п'ятого ступеня з додатного числа *a*, наближення:

$$x_{n+1}=\frac{4}{5}x_n+\frac{a}{5x_n^4}.$$

В даному випадку похибка $(n+1)$-го наближення не переважає $\frac{4}{5}a|x_{n+1}-x_n|$.

Написати програму для обчислення кореня п'ятого ступеня з числа *a* з точністю до 10^{-3}, враховуючи, що

$$x_0=\begin{cases}\min(2a;0.95), & \text{ÿêùî} \quad a \le 1\\ a/5, & \text{ÿêùî} \quad 1<a\le 25\\ a/25, & \text{ÿêùî} \quad a>25\end{cases}.$$

Вказівки: значення *a* задавати самостійно, розглянувши випадки: **a)** $a \le 1$; **b)** $1 \le a \le 25$; **c)** $a \ge 25$.

Варіант 3

Задача 1: Дано дійсні додатні числа *a,b,c,d*. З`ясувати, чи можливо прямокутник зі сторонами *a,b* вмістити усередині прямокутника зі сторонами *c,d*, щоб кожна зі сторін одного прямокутника була паралельна чи перпендикулярна кожній стороні другого прямокутника.

Вказівки:

1. Виконати програму, коли:
   a) $a<c,\ b>d$; b) $a>c,\ b<d$; c) $d>a>c,\ b<c$; d) $d>b>c,\ a<c$.
2. Значення *a,b,c,d* задавати самостійно.

Задача 2: Написати програму, яка обчислює границю послідовності $\{a_n\}$,

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

прийнявши за нього таке значення a_n, з яким $|a_{n+1}-a_n|<10^{-3}$

Вказівки: на екран вивести члени послідовності від a_1 до a_n та номер члена *n*, для якого справедлива показана вище рівність.

Варіант 4

Задача 1: Написати програму для переведення номіналу ємності конденсатора *C* у скорочену форму (пФ, нФ, мкФ, Ф) за правилом:

$$C = \begin{cases} пФ, & \text{якщо } 0 < C < 10^{-9}; \\ нФ, & \text{якщо } 10^{-9} \le C < 10^{-6}; \\ мкФ, & \text{якщо } 10^{-6} \le C < 1; \\ Ф, & \text{якщо } C \ge 1. \end{cases}$$

Вказівки: На екран вивести округлене значення ємності із відповідним позначенням, наприклад, якщо ввели 0.0047, вивести 4700 мкФ.

Задача 2: Корінь рівняння знаходиться послідовними наближеннями за формулою:

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

Написати програму, яка знаходить таке значення кореня, з яким різниця між двома сусідніми наближеннями не перебільшує 10^{-3}, виходячи із початкового наближення $x_0=1$.

Варіант 5

Задача 1: Якщо сума трьох різних дійсних чисел *x,y,z* менше за одиницю, то найменше з них замінити напівсумою двох інших; в іншому випадку замінити менше з *x,y* напівсумою двох, що залишилися.

Вказівки:

1. Виконати програму при: a) $x+y+z=1$; b) $x+y+z<1$.
2. Значення *x,y,z* задавати самостійно.

Задача 2. Дано додатні дійсні числа a,x,ε. В послідовності y_1,y_2, яка утворена за законом:

$$y_0=a,\ y_i=\frac{1}{2}(y_{i-1}+\frac{x}{y_{i-1}}),\ i=1,2,....$$

Знайти перший член y_n, для якого виконується нерівність $\left|y_n^2-y_{n-1}^2\right|<\varepsilon$.

Вказівки: значення a,x задавати самостійно. Прийняти $\varepsilon=10^{-3}$.

Варіант 6

Задача 1: Написати програму для переведення номіналу опору резистора R у скорочену форму (Ом, кОм, МОм, ГОм) за правилом:

$$R=\begin{cases} Ом, & \text{якщо } 0\le R<1000;\\ кОм, & \text{якщо } 1\,000\le R<1\,000\,\,000;\\ МОм, & \text{якщо } 1\,000\,\,000\le R<1\,000\,\,000\,\,000;\\ ГОм, & \text{якщо } R\ge 1\,000\,\,000\,\,000. \end{cases}$$

Вказівки: На екран вивести округлене значення опору із відповідним позначенням, наприклад, якщо ввели 5600, вивести 5.6 кОм.

Задача 2: Обчислити безкінцеву суму із заданою точністю $\varepsilon(\varepsilon>0)$, враховуючи, що потрібна точність досягнута, якщо наступний доданок виявився за модулем менший, ніж ε.

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

Вказівки: вважати $\varepsilon=10^{-3}$.

Варіант 7

Задача 1: Написати програму для визначення номеру телевізійного каналу за введеною несучою частотою зображення в першому діапазоні метрових хвиль. Розподіл каналів за частотою приведений в таблиці 1.4.

Таблиця 1.4 – Розподіл каналів за частотою

Діапазон хвиль	48–57	58–66	76–84	85–92	93–100	МГц
Номер каналу	1	2	3	4	5	N

Задача 2: Написати програму знаходження коефіцієнтів ряду згідно з формулою:

$$a_k=\frac{k^2+10}{k^3+1-k^2}$$, де $к=1,2,3$ – номер коефіцієнту ряду

Вказівки:

1. Знаходити коефіцієнти, доки $\left|a_{k+1}-a_k\right|<\varepsilon$.
2. Вивести на екран коефіцієнти з номерами, що діляться на *3* та ці номери.
3. Підібрати таке ε, з яким кількість членів (коефіцієнтів) ряду буде лежати в межах $30\le k\le 40$.

Варіант 8

Задача 1: Задано три дійсних числа, що є довжинами відрізків. Визначити, чи можливо побудувати трикутник з такими довжинами сторін і, якщо це можливо – визначити тип трикутника: рівносторонній (рівнобедрений), прямокутний чи інший трикутник.

Задача 2: Написати програму для знаходження загальної ємності паралельного та послідовного з'єднання *n* конденсаторів.

Вказівки: Значення *n* та ємності конденсаторів вводити з клавіатури.

Варіант 9

Задача 1: Написати програму для виводу на екран середньої температури пори року виходячи з таблиці 1.5.

Таблиця 1.5 – Температура пори року

Пора року	Весна	Літо	Осінь	Зима
tcep..	+10o	+25o	+5o	-15o

Вказівки: Пору року задавати за допомогою літер: *"B"* – весна, *"L"* – літо, *"O"* – осінь, *"Z"* – зима.

Задача 2: Вирахувати значення інтегралу $J_n=\frac{1}{e}\int_0^1 x^n e^x dx$ за наступною рекурсивною формулою: $J_n=1- nJ_{n-1}, n=1,2,\ldots$

Вказівки: Початкове наближення інтеграла прийняти рівним $J_0=1-1/e$, вирахування закінчити, коли поточне значення J_i за абсолютним значенням стане менше 10^{-4}

Варіант 10

Задача 1: Ввести *n* – ціле. Якщо $n\le 0$, то видати про це повідомлення і ввести нове значення *n*. Якщо $1\le n\le 5$, то обчислити $y=a^n+ln(b^2+c^4+d)$; попередньо здійснивши введення даних *a,b,c,d*. Вивести на екран *a,b,c,d,y*. Якщо $n>5$, обчислити $y=a^n+b+c+d$, попередньо здійснивши введення даних *a,b,c,d*. Вивести на екран *a,b,c,d,y*.

Вказівки: виконати програму при: a) $n\le 0$; b) $1\le n\le 5$; c) $n>5$.

Задача 2: Знайти границю послідовності C_i, коли *i* прямує від 0 до нескінченності, де $C_i=\frac{i^2}{\sqrt{i^4+1}}$, приймаючи за нього таке значення C_i, з яким $|C_i - C_{i-1}|<10^{-3}$.

Вказівки: На екран вивести всі значення C_i та номер останнього значення.

2 ЛАБОРАТОРНА РОБОТА №2. РОБОТА ІЗ МАСИВАМИ ЧИСЕЛ

Мета роботи: отримати практичні навички по роботі з масивами; оволодіти навичками алгоритмізації та програмування структур із вкладеними циклами, способами введення та виведення матриць.

2.1 Теоретичні відомості

Перераховний тип. У мові Сі є можливість використання особливого типу, що дозволяє створювати змінні, які приймають тільки визначені значення (константи, що задаються під час створення типу). Цей тип задається перерахуванням можливих значень, що приймаються змінними даного типу. Задавання типу здійснюється словом ***enum***, після якого у фігурних дужках перераховуються значення, що приймаються даним типом:

enum [<ім'я типу перерахувань>]{<список перерахувань>}[<імена змінних>,]

Приклад:

enum *days {mon, tues, wed, thur, fri, sat, sun} p,w;*

У прикладі змінні *p, w* можуть приймати одне із семи значень дня тижня. Після задавання типу *days* можна об'являти змінні (об'єкти) даного типу, наприклад *enum days a,b,c;*

Імена, що занесені у *days*, представляють собою константи цілого типу. Перша з них автоматично встановлюється в нуль, і кожна наступна має значення на одиницю більше, чим попередня (*tues=1, wed=2 і* т.д.). Можна присвоїти константам визначені значення цілого типу (ім'ям, що не мають їх, будуть, як і раніше, призначені значення попередніх констант, збільшених на одиницю). Наприклад:

enum *days {mon=5, tues=8, wed=10, thur, fri, sat) my_week;*

Після цього *mon=5, tues=8, wed=10, thur=11, fri=12, sat=13, sun=14.*

Тип *enum* можна використовувати для задавання констант *true=1, false=0*, наприклад ***enum*** *boolean {false true} a, b;*

За відповідністю значень констант, що присвоюються змінним типу *enum*, не здійснюється.

```
void main (void)
{
 enum my_en {a, b=10, c,d} i1;
 my_en i2,i3; //тип my_en можна використовувати для об'яви змінних
 enum vit{r, l, m=10} j1,j2;
//правильні дії
 i1=a; i2=b; i3=a; j1=j2;
//некоректні дії, дії виконуються, але визивають видачу повідомлень
 i1=0; i1=555; i1=r;
//помилкові дії
// a=b; a=j1; a=i1; a=0; b=10;
}
```

Масиви. Масив – це структурований тип даних, що складається із декількох елементів одного і того ж типу. В якості типу масиву може бути використаний будь-який відомий (оголошений) тип мови Сі. Відмінність об'яви масиву від об'яви звичайної змінної полягає в наявності квадратних дужок після назви масиву. В них вказується кількість елементів у масиві.

Приклад задавання масиву:

int a[100]; char b[30]; float c[42];

В наведеному прикладі масив *"a"* складається зі 100 цілих чисел, масив *"b"* складається із 30 символів, масив *"c"* складається із 42 дійсних чисел. Нумерація елементів масиву починається з нульового елементу.

Масиви бувають одномірні і багатомірні. Кількість вимірів масиву визначається під час задавання по кількості квадратних дужок після імені масиву. Приклад:

int a[10]; int b[100][30]; int c[20][78][45];

У прикладі масив *"a"* є одномірним, масив *"b"* – двомірним, масив *"c"* – трьохмірним. Одномірний масив іноді називають вектором. Найбільш часто використовуються одномірні і двомірні масиви.

У одномірного масиву всі елементи зберігаються у вигляді рядка. Звертання до масиву відбувається за його ім'ям. Для звертання до конкретного елементу масиву необхідно, крім імені масиву, задати порядковий номер елементу у масиві (індекс). Наприклад, звертання до дев'ятого елементу запишеться так: *a[8]*.

Двомірний масив представляється як одномірний масив, елементи якого теж масиви. Елементи двомірного масиву зберігаються по рядках. Якщо проходити послідовно по елементах масиву в порядку їх розташування у пам'яті, то швидше всього змінюється самий правий індекс. Наприклад, звертання до дев'ятого елементу п'ятого рядка виглядає так: *a[4][8]*.

Якщо заданий опис масиву: ***int*** *a[2][3];* тоді елементи масиву *a* будуть розміщуватись у пам'яті наступним чином: *a[0][0], a[0][1] a[0][2], a[1][0], a[1][1], a[1][2]*. Ім'я масиву *a* – це вказівник константи, котра містить адресу його першого елементу (для нашого прикладу – *a[0][0]*). Припустимо, що *a=1000*. Тоді адреса елементу *a[0][1]* буде дорівнювати *1002* (елемент типу ***int*** займає у пам'яті 2 байти), адреса наступного елементу *a[0][2] – 1004* і т.д. Якщо вибраний елемент, для якого не виділена пам'ять, компілятор не слідкує за цим. В результаті програма буде працювати не вірно. В ряді випадків відбудеться аварійне завершення програми.

Мова С дозволяє ініціалізувати масив під час задавання. Для цього використовується така форма:

<тип> <ім'я_масиву>[<кількість_елементів>]={<список_значень>};

Приклад:

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

В останньому випадку: *b[0][0]=1, b[0][1]=2, b[0][2]=3, b[1][0]=4, b[1][1]=5, b[1][2]=6.*

Приклад використання циклів для операцій з масивами. Зазвичай послідовний доступ до елементів масиву здійснюється за допомогою операторів циклу ***for***, ***while*** чи ***do while***. Проілюструємо використання оператора циклу і функції ***scanf*** для введення значень масиву ***int*** *a[3]*:

```
for(i=0;i<3;i++) scanf("%d",&a[i]);
чи
i=0;
while(i<3)
  {
   scanf("%d",&a[i]);
   i=i+1;
  }
або
i=0;
do
{
  scanf("%d",&a[i]);
  i++;
}
while(i<=3);
```

Введення елементів масиву можна здійснювати також за допомогою оператора ***cin*** (C++). Приклад:

```
for(i=0;i<3;i++) cin>>a[i];
```

Приклад виведення елементів масиву:

```
for(i=0;i<3;i=i+1) printf("%d",a[i]);
```

або

```
for(i=0;i<3;i=i+1) cout<<a[i];
```

Для використання операторів *cin* і *cout* необхідно підключити бібліотеку *<iostream.h>*.

Для можливості оперативно змінювати кількість елементів необхідно задати масив максимально можливої довжини. Перед використанням цього масиву необхідно буде визначити розмір фрагмента масиву, що використовується (визначити явно присвоєнням чи ввести з клавіатури). Розмір фрагменту, що використовується, повинен бути менше або дорівнювати розміру заданого масиву.

Вказівники. Вказівник – це адреса деякого об’єкту, через неї можна звертатися до цього об’єкту. Унарна операція *&* дає адресу змінної. Операцію *&* можна використовувати тільки до змінних і елементів масиву. Приклад присвоєння адреси змінної *"x"* змінній-вказівнику *"y"*:

```
int x,*y; y=&x;
```

Унарна операція * сприймає свій операнд як адресу деякого об'єкту і використовує цю адресу для вибірки вмісту. Приклад видобутку значення за адресою вказівника:

int *x,*y,z; y=&x; z=*y;*

Вказівники можна використовувати як операнди у арифметичних операціях. Якщо *y* – вказівник, то унарна операція *y*++ збільшує його значення. Для *y*++ – адреса наступного елементу. Вказівники і цілі числа можна сумувати. Конструкція *y*+*n* (*y* – вказівник, *n* – ціле число) задає адресу *n*-го об'єкту, на який вказує *y*. Це справедливо для будь-яких об'єктів (*int, char, float* і т.д.). Транслятор буде масштабувати прирощення адреси у відповідності з типом, що визначений у відповідності з об'яявленням (***int*** **y;* ***char*** **y;* ***float*** **y*).

Будь-який вказівник можна перевірити на рівність (==) чи нерівність (!=) зі спеціальним значенням *NULL*, яке записується замість нуля. Слово *NULL* дозволяє визначити вказівник, який нічого не адресує.

int **y; y=NULL;* ***int*** *x=5;* ***if****(y==NULL) y=*x;*

Після створення вказівника в ньому знаходиться довільне значення, тобто він посилається на будь-який фрагмент пам'яті. Якщо по тексту програми вказівник перевіряється на *NULL*, то після створення вказівника його необхідно приводити до значення *NULL*.

Зв'язок масивів і вказівників. Якщо задати: ***int*** *mas[100], *p, a;* то:

1) для масиву відводиться пам'ять в адресному просторі під *100* елементів типа *int*;

2) пам'ять відводиться під вказівник-константу з ім'ям *mas*, значенням вказівника є адреса масиву;

3) пам'ять відводиться під вказівник-змінну з ім'ям *p*.

Операція ініціалізації вказівника може здійснюватися тільки операцією присвоєння адреси деякої змінної:

p=&a;

p=&mas[0]; або *p=mas;*

або присвоєнням *p=NULL.*

Допустимо *p=0,* але не рекомендується.

Помилкою є:

a=10;

p=a; // де *p* – вказівник. Присвоєння неможливе, так як типи *int** і *int.*

p=10; // присвоєння неможливе, так як тип *int** и *const int*.

Вказівнику неможна присвоювати цілі значення, але можна додавати і віднімати вказівник і цілі числа. *p+=10*; – еквівалентно *p=p+10*; – збільшення адреси на *10**масштабний множник.

p-=2; зменшення на 2*масштаб множника.

**p+=10*; збільшує на 10 вміст комірки пам'яті, на яку посилається *p*.

Наприклад, якщо *p=mas;* то *p+=10*; еквівалентно *p=p+10* і еквівалентно присвоєнню *p=&mas[10];*

**p+=10*; еквівалентно *mas[0]=mas[0]+10;*

Якщо 2 вказівника посилаються на елементи одного і того ж масиву, то допускаються операції відношення над ними: ==; !=; <, >, і т.д.

Для вказівників, що посилаються на елементи різних масивів, результат порівняння не визначений.

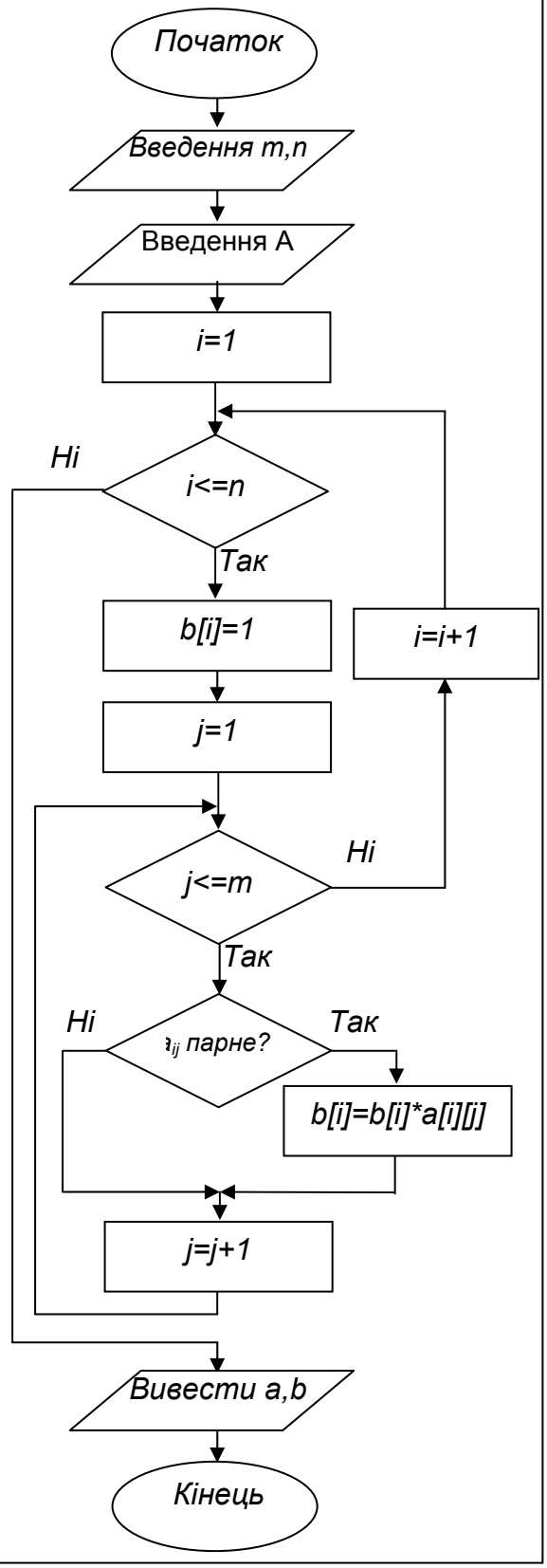
Приклад вирішення лабораторного завдання:

Задача: Дана цілочислова матриця *a* із розмірністю *m*x*n*. Отримати послідовність $b_1....b_n$, де b_i – це добуток непарних елементів в *i*-му рядку.

Вказівки: попередню матрицю задавати самостійно.

Блок-схема і текст програми:

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int a[20][20],b[20],i,j,n,m;
 clrscr();
 printf("Введіть розмірність матриці n,m:");
 scanf("%d%d",&n,&m);
 printf("Введіть матрицю a %dx%d:\n",n,m);
 for (i=1;i<=n;i++)
       for (j=1;j<=m;j++)
            scanf("%d",&a[i][j]);
 for (i=1;i<=n;i++)
 {
       b[i]=1;
       for (j=1;j<=m;j++)
            if (a[i][j]%2==0) b[i]*=a[i][j];
 }
 printf("Початкова матриця a:\n");
 for (i=1;i<=n;i++)
 {
      for (j=1;j<=m;j++)
            printf("%4d",a[i][j]);
      printf("\n");
 }
 printf("Матриця b:\n");
 for (i=1;i<=n;i++)
      printf("%d\n",b[i]);
getch();
}
```



2.2 Методичні вказівки

а) вивчити правила організації масивів та синтаксис описання в програмі одномірних та двомірних масивів(матриць);

б) вивчити організацію вкладених циклів з урахуванням порядку перебору елементів матриці;

в) вивчити правила використання прийомів програмування в структурах із вкладеними циклами;

г) розробити алгоритм вирішення задач свого варіанту, записавши їх у вигляді блок-схем;

д) забезпечити в програмі виведення коментарів та природне завершення програми в випадках можливого зациклення та зависання програми.

2.3 Контрольні запитання

1. Задавання одномірного масиву.
2. Задавання багатомірного масиву.
3. Доступ до елементів масиву.
4. Тип масиву.
5. Перерахований тип даних.
6. Вказівники.

2.4 Варіанти завдань

Варіант 1

Задача1: Помножити матрицю A_{MxN}, на вектор R, з розмірністю n за формулою:

$$U_i = \sum_{j=1}^{n} a_{ij} * r_j \quad \text{, де } i=1,2,...m,\ j=1,2,...n.$$

Вказівки: на екран вивести попередню матрицю A, вектор R (задати самостійно) та результуючий вектор U.

Задача2: Дано натуральне число n, цілочислова квадратна матриця A порядку n. Отримати $b_1...b_n$, де b_i – це найменше із значень елементів, які знаходяться на початку i-го рядка матриці до елемента, який належить до головної діагоналі включно.

Вказівки: матрицю задавати самостійно.

Варіант 2

Задача1: Знайти скласти та відняти дві матриці A і B з однаковою розмірністю mxn за формулами:

$c_{ij}=a_{ij}+b_{ij}$; $d_{ij}=a_{ij}-b_{ij}$; де $i=1,2,....m,\ j=1,2,....n$.

Вказівки: на екран вивести попередні матриці (задавати самостійно) та результуючі.

Задача2: Дана цілочислова квадратна матриця порядку *4*. Знайти найменше із значень елементів стовпця, який володіє найбільшою сумою по модулю елементів. Якщо таких стовпчиків декілька, то взяти перший з них.

Вказівки: попередню матрицю задати самостійно.

Варіант 3

Задача1: Написати програму транспонування матриці $A=(a_{ij})$, де $i=1,2,...n,\ j=1,2,...n$ для довільного значення n. Операція транспонування полягає в заміні рядків матриці стовпчиками (i-й рядок заміняється на j-й стовпчик).

Вказівки: на екран вивести попередню матрицю та результуючу (попередню матрицю задавати самостійно).

Задача2: Дано натуральне n, цілочислова квадратна матриця порядку n. Отримати $b_1....b_n$, де b_i – це сума елементів, які знаходяться після першого від`ємного елемента в $i$-му рядку (якщо всі елементи рядка невід`ємні, то прийняти $b_i=100$).

Вказівки: попередню матрицю задавати самостійно.

Варіант 4

Задача1: Помножити матрицю A з розмірністю mxn на матрицю B з розмірністю nxl за формулою

$$c_{kj}=\sum_{i=1}^{n}(a_{ki} * b_{ij}),\ \text{де } j=1,2,...l,\ k=1,2,...m.$$

Отримана матриця має розмірність mxl.

Вказівки: на екран вивести матриці A,B (задавати самостійно) та C.

Задача2: Дана дійсна матриця із розмірністю mxn. Отримати послідовність $b_1...b_n$, де b_i – це найбільше із значень елементів i-го рядка.

Вказівки: попередню матрицю задавати самостійно.

Варіант 5

Задача1: У заданій дійсній матриці із розмірністю 3x4 поміняти місцями рядок, який містить елемент із найбільшим значенням, із рядком, який містить елемент із найменшим значенням. Припускається, що такі елементи єдині.

Вказівки: попередню матрицю задавати самостійно.

Задача2: Дана дійсна матриця із розмірністю mxn. Отримати послідовність $b_1...b_n$, де b_i – це добуток квадратів тих елементів i-го рядка, модулі яких належать до відрізку [$1, 1.5$].

Вказівки: попередню матрицю задавати самостійно.

Варіант 6

Задача1: В даній дійсній квадратній матриці із розмірністю n знайти суму елементів рядка, в якій знаходиться елемент із найменшим значенням. Припускається, що такий елемент єдиний.

Вказівки: попередню матрицю задавати самостійно.

Задача2: Дана дійсна матриця із розмірністю mxn. Отримати послідовність $b_1....b_n$, де b_i – це число від`ємних елементів в i-му рядку.

Вказівки: попередню матрицю задавати самостійно.

Варіант 7

Задача 1: Написати програму для арифметичних дій над матрицею $A=[a_{ij}]$ в залежності від нажатої клавіші:

$$[A]=\begin{cases}[A]+x, & \text{якщо '+';}\\ [A]-x, & \text{якщо '–';}\\ [A]*x, & \text{якщо '*'.}\end{cases}\quad i=1,2,3...n;\ j=1,2,3...m.$$

Вказівки: m, n та початкову матрицю задати самостійно, x– вводити з клавіатури після натискання ‘+’, ’-’ або ‘*’.

Задача 2: Задане натуральне число k, цілочислова матриця порядку k. Отримати вектор-стовбець $c_1,c_2...c_i...c_k$, де c_i –сума від’ємних елементів i-го рядка.

Вказівки: значення k та початкову матрицю задати самостійно.

Варіант 8

Задача 1: Заданий двомірний масив $A=(a_{ij})$, де $i=1,2...k$, $j=1,2...f$, елементами якого є цілі числа, які складаються з будь якої кількості цифр. Написати програму для складання матриці, елементами якої будуть числа , які дорівнюють кількості цифр в однійменній комірці в масиві A.

Вказівки: початкову матрицю задати самостійно, на екран вивести початкову та результуючу матриці.

Задача 2: впорядкувати послідовність $c_1 \ldots c_n$, яка складається з дійсних чисел в порядку зменшення. Дробові числа округлити до найближчого цілого числа.

Вказівки: початкову послідовність задати самостійно, на екран вивести початкову та результуючу послідовність.

Варіант 9

Задача 1: Заданий двомірний масив $A=(a_{ij})$, де $i=1,2...n$, $j=1,2...m$,. Сформувати одномірний масив B, що складається з від'ємних елементів масиву A, та знайти їх суму.

Задача 2: Задане натуральне число k, цілочисловий одномірний масив порядку k. Поміняти у масиві максимальний елемент з першим, а мінімальний з останнім.

Вказівки: Значення k та початковий масив задати самостійно.

Варіант 10

Задача 1: Заданий двомірний масив $A=(a_{ij})$, де $i=1,2...n$, $j=1,2...m$, елементами якого є цілі числа. Впорядкувати інформацію в масиві в порядку зростання.

Вказівки:

1. Початковий масив задати самостійно.
2. На екран вивести початковий та результуючий масиви.

Задача 2: В одномірному масиві $u_1 \ldots u_n$, що складається з величин напруги джерел живлення, знайти кількість стандартних величин напруги: 1.5В, 3В, 4.5В, 6В, 9В, 12В.

Вказівки: початковий масив задати самостійно.

3 ЛАБОРАТОРНА РОБОТА №3. ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ПІДПРОГРАМ КОРИСТУВАЧА

Мета роботи: отримання навичок з алгоритмізації і програмування задач з використанням підпрограм та звертання до них; навичок з вибору параметрів підпрограм.

3.1 Теоретичні відомості

У мові С передбачено засоби, завдяки яким можна оформляти послідовність операторів як підпрограму. Підпрограма – це названа група операторів, яку можна виконати в будь-якому місці програми довільну кількість разів.

Усі підпрограми поділяються на два класи: стандартні (зарезервовані) і визначені користувачем. Стандартні підпрограми є частиною мови, вони заздалегідь не описуються. Підпрограми, визначені користувачем, обов'язково описуються. Сам опис не передбачає виконання жодних дій.

У мові С існують лише підпрограми функції, але якщо правильно описати підпрограму, то вона може працювати і як функція, і як процедура.

Функції. Функції в Сі повинні мати унікальні імена. Суттєве значення має тип значення функції, що повертається. Функція може мати параметри, що необхідні для її виконання. Список параметрів задається в круглих дужках після імені функції. Кількість параметрів може бути довільним. Найпростішим прикладом використання функції є функція ***main*** (головна), яка повинна бути присутньою в будь-якій програмі, що розроблюється на Сі. З неї починається виконання програми. Частіше всього функція *main* не має параметрів і не повертає значення. В цьому випадку замість відсутніх типів вказується ключове слово ***void*** (якщо тип параметру, що повертається, не вказаний, то по замовчуванню приймається *int*). Приклад: *void main(void).*

Якщо функція повертає значення, то це значення передається фрагменту, що її викликав, за допомогою виразу, що записується в операторі ***return***. *Return* викликає завершення роботи даної функції, передачу значення у функцію, що викликала дану і повернення керування на оператор, що йде після виклику функції.

Під час виклику функції вказуються **фактичні параметри** виклику, їх кількість повинна відповідати числу і типу **формальних параметрів** у заголовку функції, що викликається. Якщо функція не повертає значення (тобто повертає *void*, аналог процедури у паскалі), то оператор *return* може використовуватися у варіанті *«return;»* (без виразу, що слідує за ним).

Приклад:

```
#include <stdio.h>
#include <conio.h>
float pi(void)
{                              // функція тільки повертає значення константи Пі
 return 3.14159265359;         // значення що повертається
```

```
}
int SummaAandB(int A,int B)
{                                   // функція повертає суму змінних A і B
 return A+B;                        // значення що повертається
}
void PrintName(void)
{                    //функція виводить на екран ім'я Serg 10 раз по одному
                          //в рядку, не повертає і не приймає ніяких значень
for(int t=0;t<10;t++)
       printf("Serg\n"); }
void PrintA(int A)
{                           // функція виводить на екран значення змінної A
 printf("%d",A); }
void main(void)
{ float f=pi();                                 // виклик функції pi
 printf("pi=%f\n",f);
 int i=SummaAandB(1,3);             // виклик функції SummaAandB
 printf("summa(A+B)=%d\n",i);
 PrintName();                       // виклик функції PrintName
 PrintA(20);                    // виклик функції PrintA
}
```

Функції на мові Сі необхідно оголошувати (прототип функції). У відповідному оголошенню буде дана інформація про параметри. Вона представляється у наступному вигляді:

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

Для кожного параметру можна вказати тільки його тип (наприклад, <тип> <ім'я функції> (*int*, *float*);), а можна дати і ім'я параметру (наприклад, <тип> <ім'я функції> (*int a,float b*);). Дозволено створювати функції зі змінною кількістю параметрів. Тоді під час задавання прототипу замість останнього із параметрів вказується багатокрапка (наприклда *void f1(int a, float b,…).*

Можна визначити і функцію *void f1 (...) {тіло функції}* і звертання: *f1 (a1, a2)*, де *a1* и *a2* достаються якимось особливим чином із стеку чи відомі як глобальні).

Локальні і глобальні змінні. У мові Сі змінні діляться на глобальні і локальні.

Глобальні змінні об'являються у файлі початкового тексту програмного модулю поза будь-якою функцією (локальні об'являються всередині функції). Глобальні змінні створюються в точці оголошення і доступні (видимі) у початковому тексті від точки оголошення до кінця файлу, в якому вони задані (вони видимі і всередині функцій). Глобальні змінні видимі також і для зовнішніх модулів.

Локальні змінні по відношенню до функцій є внутрішніми. Вони починають існувати в точці оголошення всередині функції і знищуються після виходу з неї. Якщо вони записані в списку параметрів функції (у круглих

дужках), то треба розглядати таке оголошення як введення до першої відкриваючої фігурної дужки. Для тих локальних змінних, яких нема в списку параметрів, оголошення робиться після першої відкриваючої фігурної дужки.

В середовищі Borland C++ оголошення можна записати в будь якому місті програмного коду функції. Змінна, що оголошена у функції, буде видимою від точки оголошення до кінця блоку операторів (закриваючої фігурної дужки), в якому вона оголошена.

Програма в Сі, зв'язок між функціями і передача параметрів до функції. Програми на мові Сі зазвичай складаються з деякої кількості окремих функцій (підпрограм), серед яких повинна бути одна з ім'ям ***main,***. з якої починається виконання програми. Як правило, функції мають невеликі розміри, і можуть знаходитись як в одному, так і в декількох файлах. Якщо функції розташовані в різних фізичних файлах, то для виконання їх як одної програми, необхідно зібрати їх у файлі проекту. У Сі заборонено визначати одну функцію всередині іншої, тому всі імена функцій є глобальними. Зв'язок між функціями здійснюється через аргументи, значення, що повертаються і зовнішні (глобальні) змінні. Передача значення (повернення значення) із викликаної функції у ту, що викликала, реалізується за допомогою оператора повернення, який має вигляд: ***return*** <вираз>;

Таких операторів у підпрограмі може бути декілька, і тоді вони фіксують відповідні точки виходу. Функція, що викликає може за необхідністю ігнорувати значення, що повертається. Після слова *return* можна нічого не записувати. В цьому випадку ніякого значення не повертається. Керування передається функції, що викликала і у випадку виходу "по кінцю" без використання *return* (після закриваючої фігурної дужки).

Приклади:

```
#include <stdio.h>
int f1(void)
{ printf("працює f1()");
  return 1; }              //функція повертає значення 1
void main (void)
{ int k=f1();              //значення, що повертається використовується
}                          //у зовнішній програмі
#include <stdio.h>
int f1 (int a,int b)
{return a+b;}              //функція приймає параметри і повертає значення
void main(void)
{int a=17; int b=16;
printf("%d",f1(a,b));      //значення, що повертається, виводиться у main
}
#include <stdio.h>
f1 (int a,int b)           //функція приймає параметри
{a+b;}                     //і повертає значення за замовчуванням
void main(void)
```

```
{int a=17; int b=16;printf("%d",f1(a,b));}
                    //результат роботи такий же, як і в попередньому випадку
                    //хоча програма некоректна, про що видається
                    //попередження, як і у варіанті рядка:
{int a=17,b=16, c=f1(a,b); printf("%d",c);}
```

Приклад використання глобальних змінних:

```
#include <stdio.h>
#include <conio.h>
int a,b;                    //глобальні змінні
int f1(int x)       //x – локальний формальний параметр, приймає значення
{return (a+b)*x;}   //фактичного параметра k під час виклику функції f1(k);
void main (void)
{clrscr();
int k=35; a=10;b=12; int c= f1(k);
printf("%d",c); }                                   //c і k – локальні у main
```

Якщо деякі змінні, константи, масиви, структури оголошені як глобальні, то їх не треба включати в описок параметрів викликаної функції. Вона все рівно отримає до них доступ.

Програмі *main* також можуть бути передані параметри під час запуску. В цьому випадку текст програми може виглядати наступним чино:

```
#include <stdio.h>
void main(int argc, char *argv[])
                    //тут в argc – кількість елементів у рядку запуску програми;
{char *s; s=argv[0];      // argv[] – масив посилань на елементи цього рядка
printf("name programm=\"%s\"\n",s);       //перший елемент рядка запуску
            //конструкція \" використовується для виводу символу лапки «“»
if(argc > 1) {s=argv[1];
 printf("parametr programm=\"%s\"\n",s);} //другий елемент рядка запуску
}
```

Якщо створити програму, наприклад з ім’ям *prog.exe*, в каталозі D:\BC\WORK и запустити його із цього каталогу командою, наприклад: *prog 77*, то в результаті отримаємо виведення на екран двох рядків:

```
name programm=” D:\BC\WORK\PROG.EXE”
parametr programm=”77”.
```

Відповідність між формальними і фактичними параметрами. У мові С аргументи функції передаються за значенням, тобто функція отримує в іменах формальних параметрів тимчасову копію кожного аргументу, а не його адресу (параметри значення). Це означає, що в самій функції не може змінюватися значення самого оригінального аргументу у програмі, що викликала функцію. Якщо в якості аргументу функції використовується ім’я масиву, то передається початок масиву (адрес початку масиву), а самі елементи не копіюються. Функція може змінювати елементи масиву, зсовуючись (індексуванням) від його початку.

Відповідність між параметрами. Якщо виклик функції реалізований наступним чином:

```
int a,b=73, c=40;
    a=fun(b,c);
```

Тут *b* і *c* – параметри, значення яких передаються до підпрограми. Якщо опис функції починається так: ***void fun(int b,int c)***; то імена аргументів, що передаються у виклику і в програмі *fun* будуть однаковими. Якщо ж опис функції починається, наприклад, рядком ***void fun(int i,int j);*** , то замість ім'я *b* у функції, що викликала, для того ж аргументу у функції *fun* буде використано ім'я *i*, а замість *c – j* .

Якщо до функції звернутись так ***c=fun(&b,&c);,*** то підпрограмі передаються адреси змінних *b* і *o*. Тому прототип (заголовок функції) повинен бути, наприклад, таким: ***void fun(int *k, int *c);*** В цьому випадку *k* отримує адресу змінної *b*, а *c* – адресу змінної *c*. В результаті у програмі, що викликає, *c* – це змінна цілого типу, а в програмі, що викликана, *c* – це вказівник на змінну цілого типу. Якщо у виклику записані ті ж імена, що і в списку параметрів, але вони записані в іншому порядку, то все рівно встановлюється відповідність між *i*-м іменем у списку і *i*-м іменем у виклику.

Вище вже вказувалось, що змінні передаються функції по значенню, тому немає прямого способу у функції, що викликається, змінити деяку змінну у функції, що викликає. Однак це легко зробити, якщо передавати у функцію не змінні, а їх адреси.

Приклад:

```
#include <stdio.h>
void f1(int *a,int b, int *c1)// аргументи: a – за адресою, b – за значенням,
{*c1=*a+b; return;}      //c1 – за адресою.  Вказівник c1 посилається на c
void main (void)         // і значення c=*c1;
{int c,a=17, b=16;
f1(&a,b,&c);             //параметри, що передаються; b – за значенням
printf("%d",c);     }    // a і c – за адресою. Результат отримуємо у c.
```

Передавання масиву у вигляді параметру. Тут можливі три варіанти. Причому в прототипі і тілі записи можуть комбінуватися із цих варіантів.

1.Параметр задається як масив (наприклад: *int mn[100])*.

2.Параметр задається як масив без вказування його розміру (наприклад: *int m[ ]*).

3.Параметр задається як вказівник (наприклад: *int *m*, прототип *int f1(int *mn)*.

Незалежно від вибраного варіанту, функції, що викликана, передається вказівник на початок масиву. Самі ж елементи масиву не копіюються.

```
#include <stdio.h>
void f1 (int x[20],int k)
{int i; for (i=0; i<k; i++) x[i]=i;}
#define k 5
void main (void)
```

```
{ int a[k],i; f1(a,k);
for (i=0; i<k; i++) printf("%d;",a[i]); }
```

Функція, що викликається, може бути записана і у вигляді:

```
void f1 (int x[], int k)
{int i; for (i=0; i<k; i++) x[i]=i;}
```

або

```
void f1(int *x,int k)
{int i; for (i=0; i<k; i++) x[i]=i;}
```

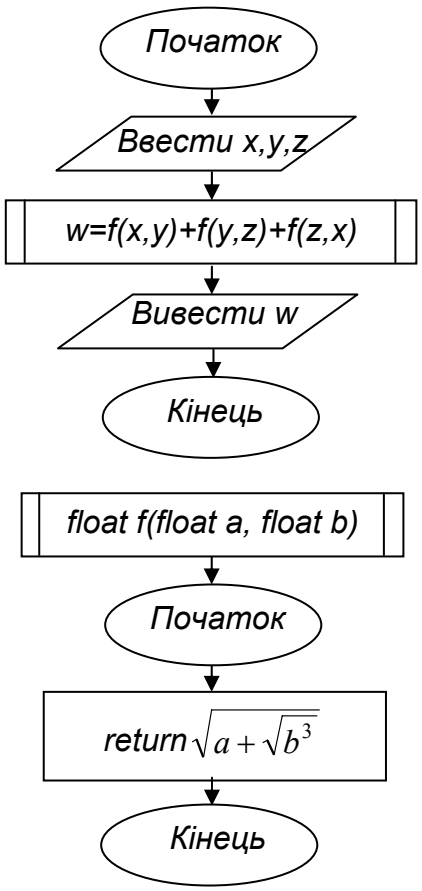
Приклад вирішення лабораторного завдання:

Задача: Скласти програму для обчислення значення функції, використовуючи підпрограму:

$$w = \sqrt{x+\sqrt{y^3}} + \sqrt{y+\sqrt{z^3}} + \sqrt{z+\sqrt{x^3}}.$$

Блок-схема і текст програми:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
float f(float a, float b)
{
 return sqrt(a+pow(b,3/2));
}
void main()
{
 float x,y,z,w;
 clrscr();
 printf("Введіть значення x,y,z:");
 scanf("%f%f%f",&x,&y,&z);
 w=f(x,y)+f(y,z)+f(z,x);
 printf("w=%f",w);
 getch();
}
```



3.2 Методичні вказівки:

а) вивчити правила запису підпрограм різних типів та засобів звертання до них;

б) вивчити засоби передачі параметрів до підпрограми ;

в) засвоїти правила запису програм, які використовують підпрограми різних видів;

г) розробити алгоритми розв`язання задач свого варіанту, записавши їх у вигляді блок-схем;

д) підготувати текстовий варіант програми та попередніх даних.

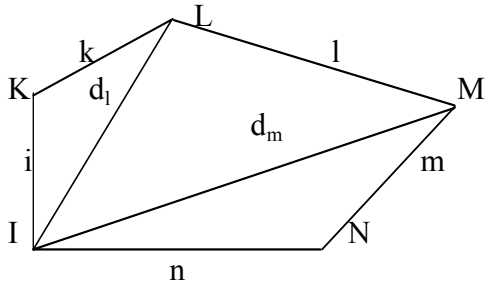
3.3 Контрольні запитання

1. Використання підпрограм-функцій у програмі.
2. Формальні та фактичні параметри.
3. Передача параметрів до функції і повернення значень.
4. Локальні та глобальні змінні.
5. Особливості складання блок-схеми програми, що має підпрограми.
6. Рекурсія.

3.4 Варіанти завдань

Варіант 1

Задача1: Написати програму знаходження кутів багатокутника *IKLMN* зі сторонами *i,k,l,m,n* та діагоналями d_l d_m.



Вказівки: кут *A* трикутника виражається через його сторони *a,b,c* за формулою: $A=2\text{arctg}\frac{(p-b)(p-c)}{p(p-a)}$, де $p=\frac{a+b+c}{2}$ – напівпериметр. Попередні дані задавати самостійно.

Задача2: Обчислити наступні функції: $x=\frac{a^{\,b}+c^{\,d}}{a^{\,c}+b^{\,d}}$; $y=(x+7)^3+(x-7)^5$;

$w=\ln(x^a+y^b)$; $v=x^{\sin b}+(8-x)^{\cos b}$;

Вказівки: попередні дані задавати самостійно; операцію піднесення до ступеня оформити у вигляді функції.

Варіант 2

Задача1: Написати програму знаходження наближеного значення інтеграла $I=\int_{10}^{20}\frac{dx}{x}$, використовуючи наступну наближену формулу:

$$I=\int_{x_0}^{x_n} f(x)\,dx \approx \sum_{i=0}^{n-1}\frac{h}{2}[f(x_i)+f(x_{i+1})],\ \text{де } h=x_{i+1}-x_i$$

Вказівки: прийняти *h=0.01;* знаходження підінтегральної функції оформити у вигляді підпрограми.

Задача2: Дано дійсні додатні числа *a,b,c*. Обчислити:

$$\frac{\max(c, a+b)+\max(a, b+c)}{1+\max(a+bc, b+ac, 15)}.$$

Вказівки: попередні дані задавати самостійно; знаходження *max* оформити у вигляді аналога процедури.

Варіант 3

Задача1: Обчислити визначений інтеграл:

$I=\int_b^a \sqrt{2x+1}dx$, за формулою: $I=\frac{b-a}{b}[\mathrm{f}(a)+4\,\mathrm{f}(\frac{b+a}{b})+\mathrm{f}(b)]$.

Вказівки: попередні дані задавати самостійно; знаходження підінтегральної функції оформити у вигляді підпрограми.

Задача2: Обчислити величину:

$y=\frac{2\,\mathrm{th}(1/2)-3\,\mathrm{th}(t-1/10)}{5-\mathrm{th}(4t-1)}$, де значення гіперболічного тангенса обчислюється за формулою:

$$th(x)=\frac{\sum_{k=0}^{5} x^{2k+1}/(2k+1)!}{\sum_{k=1}^{5} x^{2k}/(2k)!}.$$

Вказівки: обчислення значень факторіала та th*(x)* оформити у вигляді аналога процедури; значення *t* задавати самостійно.

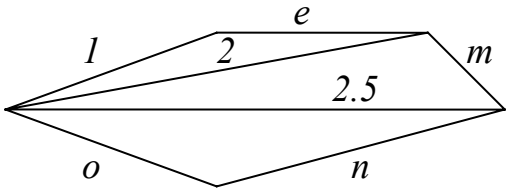
Варіант 4

Задача1: Задані дійсні числа a_1,a_2,a_3; b_1,b_2,b_3; c_1,c_2,c_3. Отримати:

$$e=\begin{cases}\min(b_1,b_2,b_3)+\min(c_1,c_2,c_3), & \text{якщо } |\min(a_1,a_2,a_3)|>10,\\ 1+[\min(\sum_{i=1}^{3}a_i,\sum_{i=1}^{3}b_i,\sum_{i=1}^{3}c_i)]^2, & - \text{ в іншому випадку.}\end{cases}$$

Вказівки: попередні дані задавати самостійно; знаходження min оформити у вигляді функції, а обчислення $\sum$ та введення a_i ,b_i, c_i– у вигляді аналога процедур.

Задача2: Задані дійсні числа *e,m,n,o*. Знайти площу п`ятикутника:



Вказівки: значення *e,m,n,o* задавати самостійно; знаходження площі трикутника за формулою Герона $S=\sqrt{p(p-a)(p-b)(p-c)}$, де *p=(a+b+c)/2* оформити у вигляді підпрограми.

Варіант 5

Задача1: По заданим дійсним числам $a_0, a_1, \ldots a_{10}; b_0, b_1, \ldots b_{10}; c_0, c_1, \ldots c_{10}; x, y, z$ обчислити величину: $\dfrac{(a_0x^{10}+a_1x^9+\cdots+a_{10})^2-(b_0y^{10}+b_1y^9+\cdots+b_{10})}{c_0(x+y)^{10}+c_1(x+y)^9+\ldots+c_{10}}$

Вказівки: попередні дані задавати самостійно; введення елементів векторів задати у вигляді процедури; знаходження ступеневого багаточлена задати у вигляді функції, яка реалізує схему Горнера: $p=a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n=(\ldots((a_0x+a_1)x+a_2)x+\ldots+a_{n-1})x+a_n)$;

Задача 2: Створити підпрограми для обчислення опору паралельного та послідовного з'єднання n резисторів. В якості параметрів підпрограм використати кількість резисторів і масив з номіналів опорів цих резисторів.

Вказівки: За допомогою створених підпрограм порахувати паралельне та послідовне з'єднання наступних резисторів:

*а)*1.5, 330, 200, 1500; *б)* 2200, 390, 1000, 4700, 56000, 750; *в)*8200, 430, 150;

Варіант 6

Задача1: Дано дійсні числа $x_1,y_1,\ldots.x_{10},y_{10}$. Знайти периметр десятикутника, вершини якого мають відповідні координати: $(x_1,y_1),\ldots.(x_{10},y_{10})$.

Вказівки: операцію знаходження довжини між двома точками, які задані своїми координатами, оформити у вигляді процедури.

Задача 2: Створити підпрограму для переведення числа із двійкової до десяткової системи числення.

Вказівки: двійкове число задавати як одномірний масив.

Варіант 7

Задача 1: Вирахувати вираз:

$$k=\begin{cases}2\cdot x^5+\sum\limits_{i=0}^{X}(2\cdot i+\dfrac{y}{i+1}), & \text{якщо } x\le 10;\\ y^{20}-x^6-\sum\limits_{i=10}^{X}(8\cdot i-\dfrac{x}{i}), & \text{якщо } x>10.\end{cases}$$

Вказівки: вирахування суми оформити у вигляді підпрограми; значення x, y вводити з клавіатури ($x>0$).

Задача 2:

Вирахувати величину $g=\dfrac{2\cdot \mathrm{k}(x,y,z)+\mathrm{k}^2(x,y,z)}{(\mathrm{k}(x,y,z)-\mathrm{k}^2(x,y,z))^2}$, де

$$\mathrm{k}(a,b,c)=\begin{cases}\dfrac{a\cdot b\cdot c}{a-b-c}, & \text{ÿêùî}\quad a>b>c;\\ \dfrac{a\cdot b\cdot c}{a+b+c}, & \text{â }{}^3\text{íøèõ}\quad \text{âèïàäêàõ}\ .\end{cases}$$

Вказівки: вирахування k*(a, b, c)* оформити у вигляді аналога процедури; значення x, y, z вводити з клавіатури.

Варіант 8

Задача 1: Вирахувати вираз: $p = \dfrac{2 \cdot x^5 + 4 \cdot x^{30} + 8 \cdot y!}{2 \cdot y + y^{15} - 25 \cdot x!}$.

Вказівки: піднесення до ступеня та знаходження факторіалу оформити у вигляді підпрограми; значення *x* та *y* вводити з клавіатури.

Задача 2: Вирахувати вираз: $y = \dfrac{2 \cdot th(\frac{1}{2}) - 25 \cdot th(t - \frac{1}{5})}{5 - th(3 \cdot t - 2)}$, де значення гіперболічного тангенсу вирахувати по формулі:

$$th(x) \approx \frac{\sum_{k=0}^{6} \frac{x^{2k+1}}{(2k+1)!}}{\sum_{k=1}^{6} \frac{x^{2k}}{2k!}}.$$

Вказівки: вираховування значення факторіалу оформити у вигляді підпрограми; вираховування значення *th(x)* оформити у вигляді аналога процедури; значення *t* вводити з клавіатури.

Варіант 9

Задача 1: Дано дійсні числа *a, b*. Знайти: $u=max(a, b)$, $v=max(ab, a+b)$, $c=max(u^5+v^5, 256)$.

Вказівки: попередні дані задавати самостійно; знаходження *max* оформити у вигляді підпрограми.

Задача2: Змінній *t* присвоїти значення *1*, якщо рівняння $x^2+6.2x+a^2=0$ та $x^2+ax+b-1=0$ мають дійсні корені та в той-же час обидва корені першого рівняння лежать між коренями другого, і присвоїти значення *0* в усіх інших випадках.

Вказівки: *a,b* задавати самостійно так, щоб перевірити усі можливі випадки. Вирішення будь-якого квадратного рівняння оформити у вигляді підпрограми-процедури.

Варіант 10

Задача 1: Задані два комплексні числа (a+ib) і (c+id) і тип операції (додавання, множення, віднімання, ділення). Створити функції для виконання арифметичних операцій над комплексними числами і вирахувати значення модулів заданих комплексних чисел.

Вказівки: *a, b, c* та *d* вводити з клавіатури.

Задача2: Дано дійсні числа *s,t*. Отримати $g(1.2, s) + g(t, s) - g(2s-1, st)$, де

$$g(a,b)=\frac{a^2+b^2}{a^2+2ab+3b^2+4}$$

Вказівки: попередні дані задавати самостійно; знаходження функції *g(a, b)* оформити у вигляді підпрограми.

4 ЛАБОРАТОРНА РОБОТА №4. ОБРОБКА СИМВОЛЬНИХ ДАНИХ

Мета роботи: отримання навичок з алгоритмізації та програмування задач, які оброблюють символьні дані; освоїти введення та виведення символьних даних, їх оброблення; навчитися використовувати стандартні процедури та функції оброблення символьних даних.

4.1 Теоретичні відомості

Рядки символів. У мові Сі нема окремого рядкового типу даних, але можливо визначити рядки двома різними способами. В першому використовується масив символів, а у другому – вказівників на перший символ масиву. Оголошення *char a[10];* вказує компілятору на необхідність резервування місця для максимум 10 символів. Константа *a* містить адресу комірки пам'яті, в якій поміщено значення першого із десяти об'єктів типу *char*. Процедури, що пов'язані з занесенням конкретного рядка у масив *a*, копіюють його по одному символу в область пам'яті, на котру вказує константа *a*, до тих пір, доки не буде скопійований нульовий символ, що обмежує рядок. Коли виконується функція типу *printf("%s",a);*, їй передається значення *a*, тобто адреса першого символу, на який вказує a. Якщо перший символ нульовий, то робота функції *printf* закінчується, а якщо ні, то вона виводить його на екран, прибавляє до адреси одиницю і знову починає перевірку на нульовий символ. Така обробка дозволяє зняти обмеження на довжину рядка (звичайно, в межах оголошеної розмірності): рядок може бути будь-якої довжини до тих пір, доки є місце в пам'яті, куди його можна помістити.

Другим способом задавання рядка є використання вказівника на символ. Об'ява *char *b;* задає змінну *b*, яка може містити адресу деякого об'єкту. Але в даному випадку компілятор не резервує місце для зберігання символів і не ініціалізує змінну b конкретним значенням. Коли компілятор зустрічає інструкцію виду *b="Москва"*, він робить наступні дії. По-перше, як і у попередньому випадку, створює в якому-небудь місці об'єктного модуля рядок *Москва*, за яким слідує нульовий символ. По-друге, присвоює значення початкової адреси цього рядку (адресу символу *M*) змінної *b*. Функція *printf("%s",b)*; працює так же, як і у попередньому випадку, виконуючи виведення символів до тих пір, доки не зустрінеться заключний нуль.

Масив вказівників можна ініціалізувати, тобто назначати його елементам конкретні адреси деяких заданих рядків під час об'яви. Якщо оголосити *char my_char [3,14]*, і ініціалізувати його даними "РОЗУМ! МАЮ Я" в пам'яті виділиться наступний блок пам'яті (рисунок 4.1).

Р	О	З	У	М	!	\0
М	А	Ю	\0	\0	\0	\0
Я	\0	\0	\0	\0	\0	\0

Рисунок.4.1 – Вигляд блока зі статичним виділенням пам'яті

Якщо ж оголосити *char * my_char [3]* і ініціалізувати тими ж словами, то в пам'яті буде виділено (рисунок 4.2), що значно економить пам'ять.

Р	О	З	У	М	!	\0
М	А	Ю	\0			
Я	\0					

Рисунок.4.2 – Вигляд блока з використанням вказівника

Приклад:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main(void)
{
 int l;
 clrscr();
 char str_array[3][10]={"Розум!","маю","я"}; //ініціалізування 2-мірного масиву
 for (l=0; l<sizeof(str_array); l++)
  printf("%c:",*(str_array[0]+l)); //виведення 2-мірного масиву символів
 // для виведення шістнадцяткових кодів 2-мірного масиву символів
 // треба замінити оператор виведення на
//printf("%x:",*(str_array[0]+l));
printf("\nДовжина 2-мірного масиву=%d\n",sizeof(str_array));
char* point_array[3]={"Розум!","маю","я"}; //ініціалізування масиву вказівників
printf("Загальна довжина масиву із вказівників=%d\n",sizeof(point_array)
+strlen(point_array[0])+strlen(point_array[1])+strlen(point_array[2]));
}
```

Операції над рядками. Бібліотека <*string.h*> (тільки частина функцій).

*char *strcat(char *s1, char *s2);* об'єднання рядків. Функція додає до рядка *s2* рядок *s1* і повертає *s1*.

*char *strncat(char *s1, char *s2, n);* те ж саме, але дописується не більше *n* символів.

*int *strcmp(char *s1, char *s2);* порівняння рядків. Функція порівнює рядки і повертає від'ємне значення, якщо *s1<s2*, нульове, якщо *s1=s2*, додатне, якщо *s1>s2*.

*int *strncmp(char *s1, char *s2, n);* те ж саме, але порівнюється *n* символів.

*char *strcpy(char *s1, char *s2);* копіювання рядків. Функція копіює рядок *s2* до рядка *s1* і повертає *s1*.

*char *strncpy(char *s1, char *s2, n);* копіює не більше *n* символів.

strlen(char s);* довжина рядка. Функція повертає довжину рядка (не включаючи символ завершення рядка *NULL*).

*char *strpbrк(char *s1, char *s2);* пошук підрядку у рядку. В разі успішного пошуку функція повертає вказівник на елемент *s1*, з якого починається *s2*, інакше повертає *NULL*.

*char *strchr (char *s, int ch)*; пошук символу в рядку. Повертає вказівник на перший елемент *s*, який співпадає з *ch*, інакше повертає *NULL*.

*char *strrchr (char *s, int ch)*; пошук символу в рядку. Повертає вказівник на останній елемент *s*, який співпадає з *ch*, інакше повертає *NULL*.

Функції для роботи з рядками з бібліотеки <*stdlib.h*>.

*double atof(const char *s);* функція перетворення з рядку *s* в тип *double.*

*int atoi(const char *s);* функція перетворення з рядку *s* в тип *int.*

*long atol(const char *s);* функція перетворення з рядку *s* в тип *long.*

*double strtod(const char *s,**end);* функція перетворює рядок *s* у числове представлення до символу, на який вказує *end*.

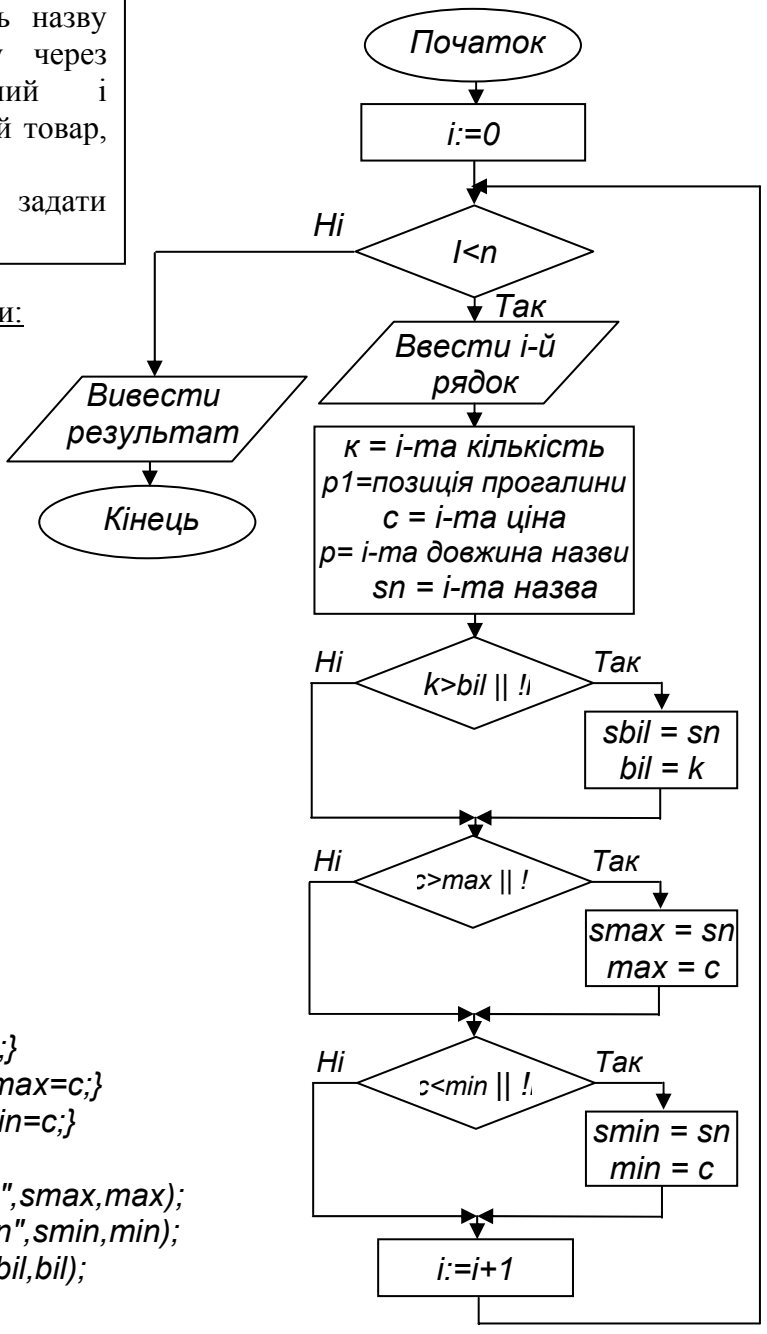
Приклад вирішення лабораторного завдання:

Задача: В масиві, який складається з n рядків, кожен з яких містить назву товару, його кількість і ціну через пробіли, знайти найдорожчий і найдешевший товари, а також той товар, якого найбільше.

Вказівки: початковий масив задати самостійно.

Блок-схема і текст програми:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
void main()
{
 const int n=5,ms=256; int i,p;
 char *s[n],*p1,smax[ms];
 char smin[ms],sbil[ms],ns[ms];
 float max,min,bil,k,c;
 clrscr();
 printf("Введіт %d рядків:\n",n);
 for (i=0;i<n;i++)
 {
  gets(s[i]);
  k=atof(p1=strchr(s[i],' '));
  c=atof(strrchr(s[i],' '));
  p=p1-s[i];
  strncpy(ns,s[i],p); ns[p]='\0';
  if (k>bil||!i) {strcpy(sbil,ns); bil=k;}
  if (c>max||!i) {strcpy(smax,ns); max=c;}
  if (c<min||!i) {strcpy(smin,ns); min=c;}
 }
 printf("Найдорожчий: %s - %g\n",smax,max);
 printf("Найдешевший: %s - %g\n",smin,min);
 printf("Найбільше: %s - %g\n",sbil,bil);
 getch();
}
```



4.2 Методичні вказівки

а) вивчити правила запису символьних даних (констант, змінних, масивів, рядків) та опис способу їх введення та виведення;

б) ознайомитися з основними стандартними функціями, які дозволяють оброблювати символьні рядки. Звернути увагу на виклик підпрограм та на типи формальних параметрів;

в) розробити алгоритми розв`язання задач свого варіанта, записавши їх у вигляді блок-схем;

г) забезпечити в програмі виведення коментарів, відповідних різним можливим ситуаціям.

4.3 Контрольні запитання

1. Задавання рядкового типу.
2. Символьний тип даних.
3. Операції з рядками
4. Операції з символами
5. Функції для дій з рядками.

4.4 Варіанти завдань

Варіант1

Задача1: Написати програму, яка реалізує епізод казки: ЕОМ питає, куди бажає піти герой (наліво, направо чи прямо), та виводить на екран, що його чекає в кожному випадку.

Вказівки: відповідь ЕОМ присвоїти символьній(рядковій) змінній та вивести на екран; текст питань та відповідей ЕОМ задавати самостійно.

Задача2: Задати інформацію для групи студентів у вигляді одномірного рядкового масиву, кожний елемент якого містить наступні данні: прізвище, ім`я, по батькові, рік народження. Визначити середній вік студентів та прізвище самого старшого.

Вказівки: результат присвоїти двом рядковим змінним та вивести на екран. (Наприклад: середній вік – 18 років. Старший – Іванов – 20 років)

Варіант 2

Задача1: У кіоску продається газета, яка коштує 1гр. та журнал, який коштує 2гр. Написати програму, яка питає про бажання покупця (журнал чи газета), приймає гроші (сума грошей вводиться з клавіатури) та виводить на екран належну здачу.

Задача2: Задати інформацію про результати сесії для групи студентів у вигляді одномірного масиву, кожний елемент якого містить інформацію про одного студента (прізвище та п`ять оцінок через кому).

Обчислити середній бал кожного студента та створити новий рядковий масив, кожний елемент якого містить інформацію у вигляді: середній бал Іванова - 4.2.

Варіант 3

Задача1: У масиві із n рядків, кожна з яких містить ім`я та прізвище, знайти тих, кого звуть Вася.

Задача 2: В одномірному масиві із *n* рядків, кожний з яких містить номінал резистора, перевести його зі скороченого вигляду у повний (МОм = $x10^6$, кОм = $x10^3$ і т.д. Наприклад: ‘3,3 ГОм’ = ‘3300000000 Ом’), а також знайти максимальний і мінімальний опір.

Вказівки: рядки та *n* задати самостійно.

Варіант 4

Задача1: Визначити, чи входить слово ”*день*” у рядок, який складається з декількох слів. Якщо так, то отримати новий рядок, який відповідає старому, але з якого виключені всі слова “*день*”.

Задача2: Задати рядковий масив, кожний елемент якого містить наступну інформацію: прізвище складальника деякого виробу даного цеху та число виробів, які той зібрав кожного дня за тиждень (через кому). Написати програму, яка видавала б значення рядкової змінної (яка визначена у програмі), що містить прізвище складальника, загальну кількість виробів за тиждень та середнє число виробів, що складалися за день (для кожного складальника). Визначити та вивести на екран прізвище того працівника, який досягнув найвищої продуктивності праці.

Варіант 5

Задача1: Написати програму, яка питає ім`я, порівнює з тими, що вона має як елементи символьного масиву та вітає або повідомляє, що “не знайома”.

Задача2: Написати програму, яка виконує наступні фінансово-економічні розрахунки. Дані у вигляді місячної заробітної плати робочих зі спеціальностями різних категорій занести в одномірний масив рядків, кожен з яких (рядок) містить: прізвище, категорію, місячний заробіток, номер цеху. Обчислити загальну суму виплат за місяць по всіх категоріях, по категоріях окремо, відсоток по категоріях від загальної суми, середню місячну заробітну плату по категоріях.

Варіант 6

Задача 1: Розробити програму, котра в текстовому рядку замінює будь-яку кількість однакових символів, що йдуть один за одним підряд на один такий же символ та цифру, яка відповідає кількості видалених символів (Наприклад:
‘*1CABk3KKK111DeFf0100fk0cccccc*’=’*1CABk3K212DeFf0101fk0c5*’).

Вказівки: початковий рядок вводити з клавіатури; отриманий рядок виводити під початковим рядком.

Задача 2: В одномірному масиві із *n* рядків, кожний з яких містить прізвище та ім’я через пробіл, знайти та вивести прізвища тільки тих, у кого ім’я складається з 4 літер.

Варіант 7

Задача1: Дано масив з *n* рядків. В кожному рядку замінити всі знаки оклику крапками.

Задача2: Написати програму, яка:

а) створює каталог виробів, які зберігаються на складі. Кожен запис каталогу є рядок і містить шифр деталі, кількість та місце знаходження. По введеному шифру ЕОМ повідомляє про кількість виробів на складі та їх місцезнаходження;

б) вносить зміни про кількість деталей, які знаходяться на складі, під час їх видачі покупцю.

Варіант 8

Задача 1: В масиві, який складається з рядків, кожен з яких містить прізвище, ім'я та по-батькові через пробіли, знайти й вивести на екран тих, у кого прізвище Іванов чи Іванова.

Вказівки: початковий масив задати самостійно.

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

Варіант 9

Задача 1: Розробити програму, котра в текстовому рядку замінює будь-яку кількість прогалин, що йдуть підряд на одну прогалину і переводить всі великі літери, крім першої, у маленькі, а цифри видаляє. (Наприклад: '*Я1СА Вк3льК1De 0100fk0 cccc*'='*Яса bкльк1de fk ccc*').

Вказівки: початковий рядок вводити з клавіатури; отриманий рядок виводити під початковим рядком.

Задача 2: В одномірному масиві із *n* рядків, кожний з яких містить номінал котушки індуктивності, перевести його зі скороченого вигляду у повний (мкГн = x10^{-6}, мГн = x10^{-3} і т.д. Наприклад: '120 мкГн' = '0.00012 Гн'), а також знайти суму всіх індуктивностей у масиві.

Варіант 10

Задача 1: В заданому рядку знайти найкоротше і найдовше слово та вказати номери позицій, з яких вони починаються.

Вказівки: початковий рядок задати самостійно.

Задача2: У продажу книг у книжковій крамниці приймає участь ЕОМ. Написати програму, яка питає у покупця назву книги, яку той бажає купити. Якщо книга є у продажу, то повідомляє, скільки та коштує, приймає гроші (сума грошей вводиться з клавіатури). Далі ЕОМ визначає належну здачу (якщо грошей внесено більше); виводить на екран "Дякую", якщо здача не потрібна; або виводить повідомлення про недостачу коштів.

Вказівки: попередні дані задавати у вигляді масиву рядків, кожний з яких містить наступну інформацію: прізвище автора, назву, ціну.

5 ЛАБОРАТОРНА РОБОТА №5. РОБОТА ЗІ СТРУКТУРАМИ ТА ФАЙЛАМИ

Мета роботи: отримання навичок з алгоритмізації та програмування задач з використанням файлових структур даних; освоїти проектування структури файлу, виведення даних до файлу та читання даних з файлу; отримання навичок з організації введення/виведення значень структурних типів даних; опанування практичними навичками програмування задач з використанням структур.

5.1 Теоретичні відомості

Структура – об'єднання одного чи більше об'єктів (змінних, масивів, вказівників, інших структур і т.д.). Як і масив вона представляє собою сукупність даних. Відмінністю є те, що до її елементів необхідно звертатися за ім'ям і що різні елементи структури можуть мати різний тип. Об'ява структури здійснюється за допомогою ключового слова *struct*, за яким іде її тип і далі список елементів, що взяті у фігурні дужки:

Struct <ім'я типу структури> {
<тип поля 1> <ім'я поля>,… ;
…….
<тип поля n> <ім'я поля>,…;} <ім'я змінної>;

Ім'ям змінної може бути будь-який ідентифікатор, що записується після заключної фігурної дужки структури. В одному рядку можна записати через кому декілька ідентифікаторів одного типу. Наприклад:

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

Після фігурної дужки, що закінчує список полів, можуть записуватися змінні даного типу, наприклад: *struct date {int day;int month; int year;} a,b,c;* В цьому випадку кожній змінній виділяється пам'ять, об'єм якої визначається сумою довжин всіх елементів структури. Опис структури без наступного списку змінних не викликає виділення ніякої пам'яті, а просто задає шаблон нового типа даних, які мають форму структури. Введене ім'я типу пізніше можна використовувати для оголошення структури, наприклад *struct date days;*. Тепер змінна *days* має тип *date*. За необхідністю, структури можна ініціалізувати, поміщуючи за оголошенням список початкових значень елементів. Дозволяється вкладені структури одна в іншу, наприклад:

```
struct man { char name [30], fam [20];
             struct date bd; int voz; }
```

Визначений вище тип *data* включає три елементи: *day, month, year*, що містять цілі значення (*int*). Структура *man* включає елементи: *name[30], fam[20], bd* і *voz*. Перші два *name[30]* і *fam[20]* – це символьні масиви із 30 і 20 елементів кожний. Змінна *bd* представлена складеним елементом (вкладеною структурою)

типу *data*. Елемент *voz* містить значення цілого типа (*int*). Тепер дозволяється оголосити змінні, значення яких належать введеному типу:

```
struct man _ man_ [100];
```

Тут оголошений масив *_man_*, що складається із 100 структур типу *man*. У мові Сі дозволено використовувати масиви структур. Структури можуть складатися із масивів і інших структур.

Щоб звернутися до окремого елементу структури, необхідно вказати її ім'я, поставити крапку і зразу за нею написати ім'я потрібного елемента, наприклад:

```
_man_ [i].voz = 16; _man_ [j].bd.day = 22; _man_ [j].bd.year =1976;
```

Під час роботи зі структурами необхідно пам'ятати, що тип елементу визначається відповідним рядком оголошення у фігурних дужках. Наприклад, *_man_* має тип *man*, *year* – є цілим числом і т.д. Оскільки кожний елемент структури відноситься до визначеного типу, його ім'я може з'являтися всюди, де дозволено використовувати значення цього типу. Допускаються конструкції виду `_man_[i] = _ man_[j];` де *_man_[i]* і *_man_[j]* – об'єкти, що відповідають одному опису структури. Другими словами, дозволено присвоювати одну структуру іншій за їх іменами.

Унарна операція *&* дозволяє взяти адресу структури. Наприклад:

```
struct date { int d,m,y; } day;
```

Тут *day* – це структура типу *date*, яка включає три елементи: *d,m,y*. Інш оголошення `struct date _*db`; встановлює той факт, що *db* – це вказівник на структуру типу *date*. Запишемо вираз: `db = &day;`. Тепер для вибору елементів *d, m,* у структури необхідно використовувати конструкції: *(*db).d, (*db).m, (*db).y. db* – це адреса структури, **db* – сама структура. Круглі дужки тут необхідні, так як крапка має більш високий, ніж зірочка, пріоритет. З аналогічною метою у мові Сі передбачена спеціальна операція –>. Вона теж вибирає елемент структури і дозволяє представити розглянуті вище конструкції в більш простому вигляді: *db->d, db->m, db->y.*

Суміш – це різновид структури, котра може зберігати (в різний час) об'єкти різноманітного типу і розміру . В результаті з'являється можливість роботи в одній і тій же частині пам'яті з даними різними виду. Для опису суміші використовується ключове слово *union*, а відповідний синтаксис аналогічний синтаксису структури. Приклад:

```
union r { int ir; float fr; char cr; } z;
```

Тут *ir* має розмір 2 байти, *fr* – 4 байти і *cr* – 1 байт. Для *z* буде виділена пам'ять, достатня щоб зберегти найбільший із трьох наведених типів. Таким чином, розмір *z* буде 4 байти. В один і той же момент часу у *z* може мати значення тільки одна із вказаних змінних (*ir, fr, cr*). Приклад:

```
#include <stdio.h>
union r{ int ir; float fr; char cr;} z;
float f;
//оголошена суміш z типа r. Розмір суміші буде визначатися розміром
//самого довгого елементу, в даному випадку fr,
```

```
void main (void)
{//у Borland C++ версії 3.1 виявлена помилка під час використання
// у вирахуваннях і перетвореннях виводу дійсних значень
// елементів структур. Щоб обійти помилку, вибираємо дійсне
// значення елементу union в просту дійсну змінну f (f=z.fr;),
// а потім використовуємо f у виразах і навпаки.
printf ("розмір z=%d байти \n",sizeof(z));
// sizeof(z) вираховує довжину змінної z і printf виводить цю довжину
printf ("введіть значення z.ir \n");      //видача запиту для введення
scanf ("%d",&z.ir);                  //введення цілого значення у елемент z.ir
printf ("значення ir =%d \n",z.ir);          //виведення значення z.ir
printf ("введіть значення z.fr \n");        //запрошення для введення
                                             //дійсного значення
scanf ("%f",&f);                     //введення дійсного значення у змінну f і
z.fr=f;//запис у z.fr (фактично реалізоване введення: scanf("%f",&z.ir);
printf ("значення fr=%f \n",f); //виведення значення дійсної змінної
printf ("введіть значення z.cr \n"); // запрошення на введення інформації
flushall();                          //очищення буферів введення-виведення.
//Таке очищення буферу тут необхідне, так як у буфері введення
//залишається символ кінця рядка від попереднього введення, який потім
//введеться специфікацією %c, замість реально набраного символу
scanf ("%c",&z.cr);       //читання символу з клавіатури
printf ("значення cr=%c;\n",z.cr);       //виведення значення символу
}
```

Файли. Файл – це обмежений набір даних, що розташовані на зовнішньому носію.

У файлі розміщуються дані, що призначені для тривалого зберігання. Кожному файлу дається унікальне ім’я.

У мові С відсутні інструкції для роботи з файлами. Всі необхідні дії виконуються через функції, що включені в стандартну бібліотеку. Вони дозволяють працювати з різними пристроями, такими, як диски, принтер, комунікаційні порти т.д. Ці пристрої сильно відрізняються один від іншого. Однак файлова система дозволяє перетворювати їх у один абстрактний логічний пристрій, що називається потоком. Існує два типа потоків: **текстові і двійкові**.

Перш ніж читати чи записувати інформацію у файл, він повинен бути відкритий. Це можливо зробити за допомогою бібліотечної функції *fopen*. Вона бере зовнішнє ім’я файлу (наприклад C:\MY_FILE.TXT) і зв’язує його з внутрішнім логічним ім’ям, яке використовується надалі в програмі. Логічне ім’я – це вказівник на потрібний файл. Його необхідно оголосити, і робиться це, наприклад, так:

***FILE** *lst*; Тут *FILE* – ім’я типу, що описане в стандартному визначенні *stdlo.h*, *1st* – вказівник на файл. Звертання до функції *fopen* у програмі відбувається так: *lst=fopen(ім’я файлу, вид файлу)*;

Ім'я файлу може бути, наприклад: C:\MY_FILE.TXT – для файлу MY_FILE.TXT на диску C:; A:\MY_DIR\EX2_3.CPP – для файлу EX2_3.CPP у піддиректорії A:\MY_DIR і т.д. Вид файлу може бути:

r – відкрити існуючий файл для читання; *w* – створити новий файл для запису (якщо файл з вказаним ім'ям існує, то він буде переписаний) *a* – доповнити файл (відкрити існуючий файл для запису інформації, починаючи з кінця файлу, або створити файл, якщо він не існує); *pb* – відкрити двійковий файл для читання; *wb* – створити двійковий файл для запису; *ab* – доповнити двійковий файл; *rt* – відкрити текстовий файл для читання; *wt* – створити текстовий файл для запису; *at* – доповнити текстовий файл; *r+* – відкрити існуючий файл для запису і читання; *w+* – створити новий файл для запису і читання; *a+* – доповнити або створити файл з можливістю запису і читання; *r+b* – відкрити двійковий файл для запису і читання; *w+b* – створити двійковий файл для запису і читання; a+b – доповнити двійковий файл з наданням можливості запису і читання.

Якщо режим *t* або *b* не заданий (наприклад, *r*, *w* або *a*), то він визначається значенням глобальної змінної *_fmode*. Якщо *_fmode = 0_BINARY*, то файли відкриваються у двійковому режимі, а якщо *_fmode =O_TEXT* – у текстовому режимі. Константи *0_BINARY* і *O_TEXT* визначені у файлі *fcntl.h*. Рядки виду *r+b* можна записувати і в іншій формі: *rb+*. Якщо в результаті звертання до функції *fopen* виникає помилка, то вона повертає вказівник на константу NULL. Після закінчення роботи з файлом, він повинен бути закритий. Це робиться за допомогою бібліотечної функції *fclose*. Вона має наступний прототип: *int fclose(FILE *lst);*

При успішному завершенню функція *fclose* повертає значення нуль. Будь-яке інше значення говорить про помилку.

Виведення інформації до файлу

```
#include <stdio.h>
void main (void) {
char str[50];
FILE *rstr, *wstr, *pstr, *astr;
 rstr =fopen("c:\\my_file.txt","rt");
 wstr =fopen("c:\\out_file.txt","wt");
 pstr =fopen("prn","wt");
 astr =fopen("c:\\out_plus.txt","at");
while (fscanf(rstr, "%s", str)!=EOF)
{
 printf("Виведення на дисплей: %S\n", str);
 fprintf(wstr, "%s\n", str)   ;//запис файлу (попередній вміст витирається)
 fprintf (pstr, "%s\n", str);  // виведення на друк
 fprintf (astr, "%s\n", str);  //доповнення файлу
}
fclose(rstr); fclose(wstr); fclose(pstr); fclose (astr);
}
```

В даному прикладі вказівники не ініціюються адресами відповідних файлів, що відкриті для вказаного типу операцій. Ім'я "prn", що використовується для виведення на друк, є стандартним ім'ям друкуючого пристрою.

Читання рядків із файлу і виведення їх на екран

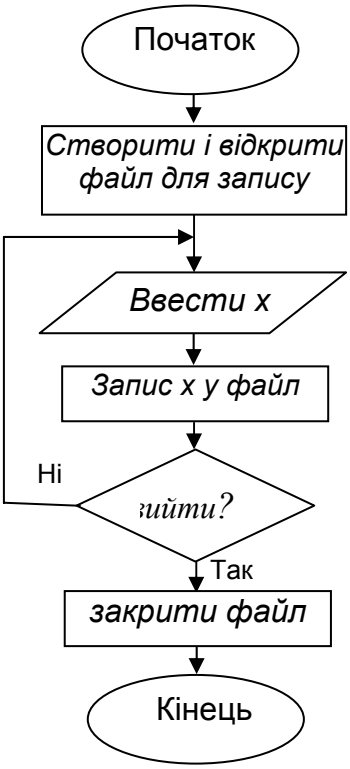
```
#include <stdio.h>
void main (void)
char  str [50];
FILE * fr, * fw;
if ((fr=fopen ("A:\\fail.ttt","r+" ))==NULL)
//відкривання файлу з дискети
{
printf("Файл не відкрився. \nВведіть інформацію з клавіатури");
fgets(str,49,stdin);                                    // можна gets (str ,49);
}
else
fgets (str,49,fr);                                      //або введіть рядок до 49
printf ("Виведення рядку: %s", str);                    //символів без прогалини
if ((fw=fopen("a:\\1.txt","w+"))==NULL)
{
 printf("Файл не відкрився");
}
else
{
 printf("\n у файл 1.txt ");                            //запис до файлу
 fputs(str,fw);                                         //функція записує
return;                                                 //вихід із програми
}       // якщо файл не відкрився, то виведення із str до файлу помилок
fprintf(stderr,"Виведення в стандартний файл помилок\n%s",str);
fclose(fr);
fclose(fw);
 }
```

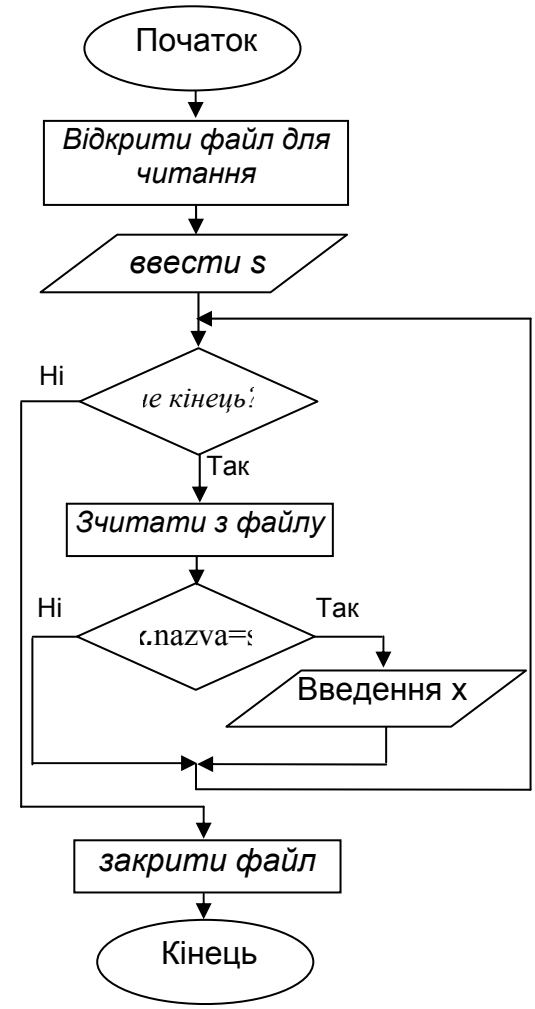
Програма зчитує із файлу *fail.ttt* дискети, в дисководі A: 49 символів або доки не зустрінеться символ кінця рядку. Якщо файл не відкрився, то пропонує ввести інформацію з клавіатури (вводиться 49 символів або до натискання клавіші ENTER). Потім інформація виводиться у файл 1.txt на дискеті або, якщо не вдалось його відкрити, у файл помилок на екрані.

Приклад вирішення лабораторного завдання:

Задача А: Створити програму для запису у файл асортименту товару в магазині. Структура запису: назва товару, кількість, ціна.

Задача В: знайти у файлі інформацію про товар, назва якого вводиться з клавіатури.





Блок-схеми і текст програм:

Задача А:

```
#include <stdio.h>
#include <conio.h>
int main(void){
struct tov{char nazva[255];int kilk;float cina;} x;
FILE *f;
clrscr();
if ((f=fopen("market.dat","wb"))==NULL)
  {fprintf(stderr,"Cannot open output file.\n");
   getch(); return 1;}
do {printf("Введіть інформацію про товар\n");
     printf("назва:");scanf("%s",&x.nazva);
     printf("кількість:");scanf("%i",&x.kilk);
     printf("ціна:");scanf("%f",&x.cina);
    fwrite(&x,sizeof(x),1,f);
    printf("Для виходу натисніть ESC\n");
   }
while (getch()!=27);
fclose(f);return 0;
}
```

Задача В:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main(void){
struct tov{char nazva[255];int kilk;float cina;} x;
char s[255];
FILE *f;
clrscr();
if ((f=fopen("market.dat","rb"))==NULL)
  {fprintf(stderr,"Cannot open  file.\n");
   getch(); return 1;}
printf("Введіть назву товару\n");
scanf("%s",&s);
while (fread(&x,sizeof(x),1,f)!=0)
if (strcmp(s,x.nazva)==0)
     printf("кількість: %d, ціна: %g",x.kilk,x.cina);
fclose(f);getch();return 0;
}
```

5.2 Методичні вказівки

а) вивчити основну термінологію, яка пов`язана з файловими структурами даних: файл та його структура, фізичний та логічний записи, методи доступу, поточний вказівник файлу;

б) вивчити стандартні функції, які забезпечують основні операції з бінарними та тестовими файлами;

в) засвоїти правила роботи зі структурами та сумішами

г) розробити алгоритми розв`язання задач свого варіанта, записавши їх у вигляді блок-схем;

д) підготувати текстовий варіант програми та попередніх даних. Провести відлагодження програми, перевіривши всі можливі ситуації.

5.3 Контрольні запитання

1. Тип даних структура.
2. Тип даних об’єднання (суміш).
3. Файловий тип даних.
4. Операції з файлами
5. Особливості роботи з файлами у програмі.

5.4 Варіанти завдань

Варіант1

Задача А: Створити файл, який містить інформацію про особисту колекцію книголюба. Структура запису: шифр книги, автор, рік видання, місцезнаходження (номер стелажу, шафи та т.і.). Кількість записів довільна.

Задача В: Написати програму, яка видає наступну інформацію:

— місцезнаходження книги автора *A* назви *B*. Значення *A,B* ввести з клавіатури;

— список книг автора *C*, які знаходяться в колекції;

— кількість книг видання *X* року, які знаходяться в колекції.

Варіант 2

Задача А: Створити файл-довідник, який містить дані про біполярні транзистори. Структура запису: марка, провідність (n-p-n, p-n-p), максимальний струм колектора, максимальна напруга колектор-емітер, мінімальний і максимальний коефіцієнти підсилення (h21e), максимальна робоча частота. Кількість записів довільна.

Задача В: Написати програму, яка дозволяє шукати у довіднику:

— всю інформацію по введеній марці транзистора з клавіатури;

— по введеному з клавіатури струму, напрузі і коефіцієнту підсилення видати всі підходящі транзистори;

— видати всі комплементарні пари транзисторів (у яких параметри однакові, а провідність різна).

Варіант 3

Задача А: Утворити файл, який містить інформацію про співробітників університету. Структура запису: прізвище працюючого, назва відділу, рік народження, стаж роботи, посада, оклад. Кількість записів довільна.

Задача В: Написати програму, яка видає дозволяє отримати наступну інформацію:

— список працівників пенсійного віку на сьогоднішній день з зазначенням стажу роботи;

— середній стаж працюючих у відділі *X*.

Варіант 4

Задача А: Утворити файл, який містить інформацію про пацієнтів дитячої клініки. Структура запису: прізвище пацієнта, стать, вік, місце проживання (місто), діагноз. Кількість записів довільна.

Задача В: Написати програму, яка видає наступну інформацію:

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

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

Варіант 5

Задача А: Утворити файл, який містить інформацію про здачу студентами сесії. Структура запису: індекс групи, прізвище студента, оцінки з п`яти екзаменів та п`яти заліків (“*З*” – зараховано, “*Н*” – не зараховано). Кількість записів довільна.

Задача В: Написати програму, яка видає наступну інформацію:

— прізвища невстигаючих студентів з вказівкою індексів груп та кількостей заборгованостей;

— середній бал, отриманий кожним студентом групи *X*, та всією групою в цілому.

Варіант 6

Задача А: Утворити файл, який містить інформацію про асортимент взуття в крамниці фірми. Структура запису: артикул, назва, кількість, ціна однієї пари. Кількість записів довільна. Артикул починається з літери *Ж* для жіночого взуття, *Ч* – чоловічого, *Д* – дитячого.

Задача В: Написати програму, яка видає наступну інформацію:

— про наявність та ціну взуття артикула *X*;

— асортиментний список жіночого взуття з вказівкою назви та кількості пар кожної моделі, яка є у продажу.

Варіант 7

Задача А: Створити файл-довідник, який містить дані про напівпровідникові діоди. Структура запису: марка, максимальний струм,

максимальна зворотна напруга, падіння напруги у відкритому стані, максимальна робоча частота. Кількість записів довільна.

Задача В: Написати програму, яка дозволяє шукати у довіднику:

— всю інформацію по введеній марці діода з клавіатури;

— по введеному з клавіатури струму, зворотній напрузі і частоті видати всі підходящі діоди;

— видати всю інформацію про діоди з падінням напруги у відкритому стані менше, ніж введено з клавіатури.

Варіант 8

Задача А: Створити файл, який містить значення функції sin(x), cos(x), tg(x) коли *x* змінюється від 0 до 314 з кроком 0.5.

Задача В: Написати програму, яка у файлі, шукає від'ємні елементи, і коли вони є, то виводить їх на екран. Коли від'ємних елементів немає, на екран вивести перший та останній елементи.

Варіант 9

Задача А: Створити файл, який містить інформацію про наявність квитків і рейсів Аерофлоту. Структура запису: номер рейсу, пункт призначення, час вильоту, час прибуття, кількість вільних місць у салоні. Кількість записів довільна.

Задача В: Написати програму, яка видає інформацію наступного типу:

— час відправлення літаків у місто *X*;

— наявність вільних місць на рейс у місто *X* ;з часом відправлення *Y*.

Вказівки: значення *X,Y* вводиться по запиту з клавіатури.

Варіант 10

Задача А: Написати програму яка створює файл, що містить інформацію про розклад телепрограм на день. Структура запису: назва програми, час початку програми, час закінчення програми.

Задача В: Написати програму, яка дозволяє отримати наступну інформацію:

— усю програму телепередач на день;

— по введеному з клавіатури часу видати назву програми, котра буде транслюватися в цей час;

— назву самої довгої та самої короткої (за тривалістю) телепрограми.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Проценко В.С. та ін. Техніка програмування мовою Сі : Навч. Посібник / В.С. Проценко, П.Й. Чаленко, А.Б. Ставровський. К.: Либідь, 1993. – 224с.
2. А.Н. Маслов. Введение в язык программирования С. М.: 1991.
3. Берри Р., Микииз Б. Язык Си: Введение для программистов / Пер. с англ. и предисл. Д.Б. Подшивалов. – М.: Финансы и статистика, 1988. – 191с. ил.
4. Романовская Л.М. и др. программирование в среде си для ПЭВМ ЕС / Л.М. Романовская, Т.В. Русс, С.Г. Свитковский. М.: Финансы и статистика, 1991. – 352с. ил.
5. Подбельский В.В. Язык Си++; Учеб. пособие, – 5-е изд. М.: Финансы и статистика, 2003. – 560 с.: ил.
6. Франка П. С++: Учебный курс. – СПб.: Питер: 2003. – 521 с.: ил.

ДОДАТОК А – РОБОТА В СЕРЕДОВИЩІ BORLAND C++

Інтегроване середовище (Integrated Development Environment – IDE) – це програма, яка складається з вбудованого текстового редактора, підсистеми роботи з файлами, систему допомоги, вбудований відлагоджувач, підсистему керування компілюванням і редагуванням зв'язків, компілятор і редактор зв'язків, бібліотеки заготовочних файлів (з розширенням *.h) а також програми-утіліти. IDE дає можливість отримати EXE-файл, не використовуючи інші програми. Середовище запускається файлом **BC.EXE**. Після запуску на екрані відображується основне вікно IDE (рисунок А.1).

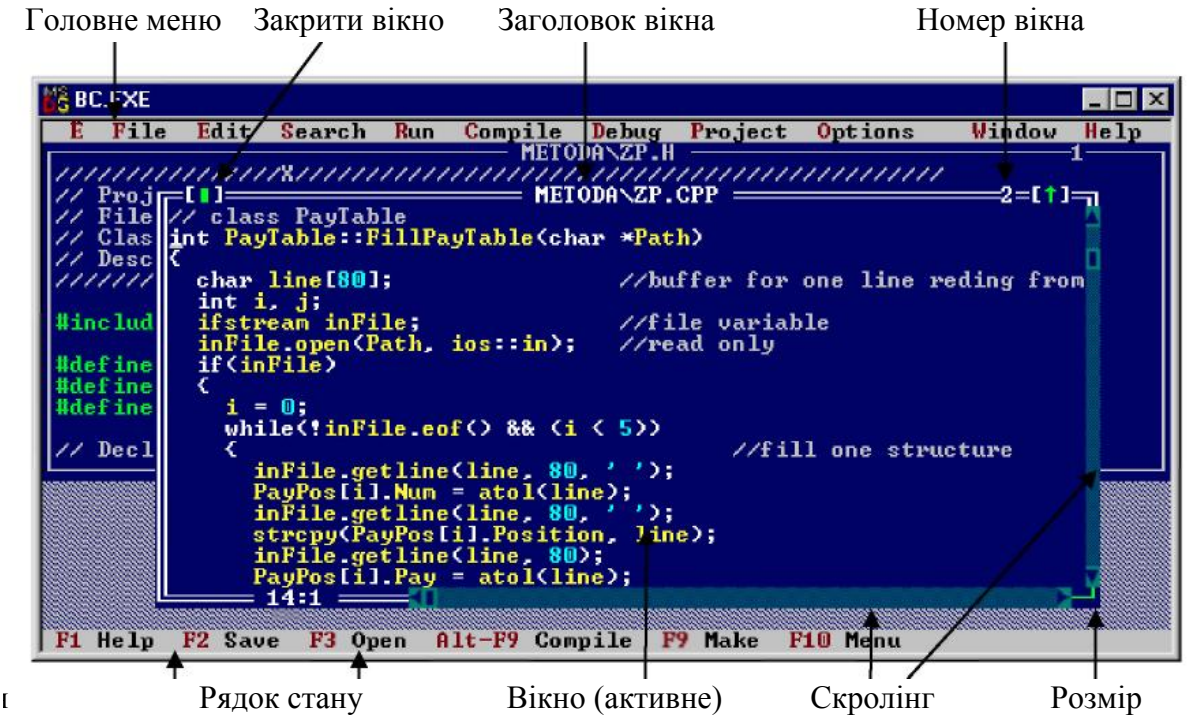


Рисунок А.1 – Основне вікно Borland C++ з двома відкритими файлами

Верхній рядок вікна – це головне меню. Опції меню дозволяють звернутися до підменю і вибрати відповідну команду.

Нижній рядок екрану відведений під рядок стану, де виділені призначення «гарячих» клавіш, в поточному режимі роботи.

Вибрати будь-яку із команд меню можна одним із трьох способів:

1)нажати клавішу F10 і за допомогою клавіш зі стрілками вибрати необхідну команду; 2)встановити курсор мишки на будь-яке ключове слово меню і натиснути ліву кнопку мишки; 3)використовувати «гарячі» клавіші (метод швидкого виклику команди). Одночасне натискання клавіші Alt і «гарячої» (буква, що виділена іншим кольором).

Активне вікно позначається подвійною лінією. В будь-який момент тільки одне вікно може бути активним. Кожне вікно має заголовок і номер, що показані в його верхньому рядку. Для активного вікна там же розташовані два керуючих поля у квадратних дужках: поле з правої сторони використовується для розкриття вікна мишкою на повний екран, а поле з лівої сторони – для закривання вікна.

Операції з вікнами можуть виконуватися трьома способами: 1)через команди головного меню **Window**; 2)за допомогою маніпулятора мишка; 3)за допомогою «гарячих» клавіш.

Для закривання активного вікна вибирається в меню *Window* команда *Close*, або нажимається «гаряча» клавіша Alt+F3, або мишкою вибирається поле [■] на вікні.

Переключення між вікнами виконується так. Вибирається команда *List All*... із меню *Window* або натискається «гарячая» клавіша Alt-0. Циклічний перегляд вікон можливий за допомогою клавіші F6 (Shift+F6).

Активне вікно може бути розкрите на весь екран або вибором *Window – Zoom*, або натисканням клавіші F5, або вибором мишкою поля [↑]. Для перегляду результатів виконання програми, якщо виведення виконується у текстовому режимі, використовується переключення у вікно виводу (*Window – Output*). Для перегляду результатів як у текстовому, так і графичному режимах, необхідно активізувати вікно екрану користувача (*Window – User screen*) або скористатися комбінацією клавіш – Alt+F5. повернення в середовище відбувається під час натискання будь-якої клавіші.

Переключення у режим редагування відбувається автоматично під час вибору команди *New* у меню *File* або під час відкривання файлу. Для повернення із меню в режим редагування достатньо натиснути клавішу Esc.

Команди вставляння і видалення: Ins – режим вставляння/заміни; Del – видалити символ в позиції курсору; Backspace – видалити символ з ліва від курсору; Ctrl+Y – видалити рядок; Ctrl+N – вставити рядок.

Команди роботи з блоками: Shift+клавіші зі стрілками – виділення блока тексту; Ctrl+Ins – копіювати блок в буфер обміну; Shift+Ins – копіювати блок із буферу обміну в поточну позицію курсору; Ctrl-Del – видалити блок. Shift+Del – вирізати блок в буфер обміну.

Основні етапи створення програми в IDE Borland C++:

1)настроювання опцій середовища програмування; 2)набір початкового тексту програми; 3)компілювання програми; 4)компонування програми; 5)відлагодження програми; 6)запуск програми на виконання.

Задавання опцій інтегрованого середовища. Першим кроком під час роботи з IDE є настроювання потрібних опцій (додаткових параметрів). Всі опції мають значення за замовчуванням. Розглянемо основні опції, що настроюються за допомогою команд меню ***Options.***

Для того щоб почати роботу з IDE, перш за все треба задати директорії, що використовуються текстовим редактором, компілятором і компанувальником (рисунок А.2). Для цього використовується команда *Options – Directories*. Поле введення *Include Directories* використовується для задавання директорій заголовкових файлів. В полі введення дозволяється вказувати декілька директорій, що розділяються символом “;”. Поле введення *Library Directories* задає директорії, що містять об’єктний файл завантажувача (CO?.OBJ, де ? – це буква M, S, H, T, L, C в залежності від моделі пам’яті, що використовується) і файлів бібліотек функцій (.LIB). Поле вводу *Output*

Directory задає директорію, в яку поміщаються файли з розширенням .OBJ, .EXE, .MAP. Якщо поле пусте, то використовується поточна директорія.

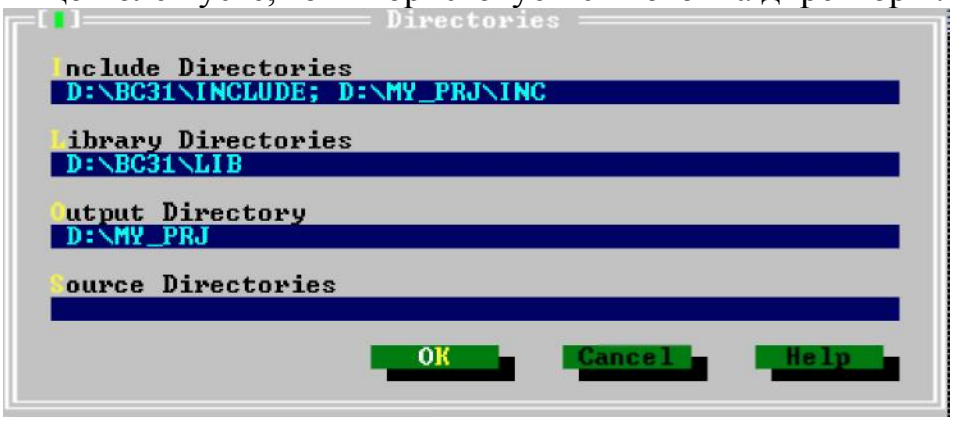


Рисунок А.2 – вікно діалогу для встановлення директорій

Під час вибору рядку *Options – Compiler* відкривається ще одне меню для настроювання опцій компілятора. Найбільш важливі опції задаються під час вибору команди *Code generation*. Опція вважається вибраною, якщо вона помічена символом (•) і включеною, якщо вона помічена символом [x]. Самим важним пунктом у вікні *Code generation* є вибір моделі пам'яті. Для більшості програм, що розроблюються для ОС MS-DOS, треба вибирати SMALL модель пам'яті.

Набор тексту програми. Наступним кроком є введення програми з використанням текстового редактора і збереження тексту програми у файлі.

Набір тексту програми можна виконати вбудованим або будь-яким іншим текстовим редактором. За традицією файли, що містять вихідний текст програм на мові С, мають розширення імені файлу .C, а на мові С++ – .CPP.

Не треба починати компілювання, компоновку чи запуск програми без збереження зробленого набору! Запущена на виконання програма може викликати «зависання» комп'ютера і зроблений набір буде втрачений.

До програм-утиліт відносять асемблер, передпроцесор, відлагоджувач, програму профілювання і багато інших корисних програм.

Компіляція, редагування зв'язків, запуск програми на виконання. Borland C++ включає найбагатші бібліотеки функцій для керування файлами, виконання введення-виведення і багатьох інший дій. Прототипи (заголовки функцій з описом типів формальних параметрів і типів значень, що повертаються), символічні константи і інш., що пов'язані з бібліотечними функціями, об'єднують у заголовочні файли, які по традиції мають розширення .H. Необхідні для компіляції файли включаються в текст програми за допомогою передпроцесорної директиви *#include*. Під час запуску на компіляцію текст програми спочатку оброблюється передпроцесором, котрий оброблює тільки «свої» директиви (наприклад, замість директиви «#include ім'я_файла» вбудовує із бібліотеки INCLUDE файл, ім'я якого задано в директиві), а потім текст програми передається безпосередньо на обробку компілятору.

Компіляція початкового тексту програми запускається або через команду *Compile – Compile to OBJ*, або натисканням «гарячої» клавіші Alt+F9. Команда *Make EXE file* також запускає програму на компілювання і за відсутності синтаксичних помилок автоматично запускає компонувальник для отримання .EXE-файлу. Ще одна можливість для запуску програми на компіляцію – команда *Run – Run* (Ctrl+F9). Після успішної компіляції і компоновки запускається отриманий .EXE-файл на виконання.

Всі повідомлення про помилки і попередження IDE поміщує у вікно з ім'ям Message. Це вікно активне після завершення компіляції. Якщо у програмі виявлені помилки, включаються засоби трасування помилок, котрі зв'язують рядки тексту програми у вікні редактора з рядками вікна Message. Переміщення висвітлення клавішами зі стрілками у вікні Message синхронно супроводжуються висвітленням відповідних помилкових рядків у тексті програми. Під час натискання клавіші Enter активізується вікно редактора і курсор встановлюється на помилковий рядок. Натискання клавіші F6 (перехід чи активізація наступного вікна) знову робить активним вікно Message.

Багатофайлове компілювання. Під час модульного програмування не обійтись без багатофайлової компіляції. Під час роботи з великими програмами набагато зручніше розміщувати частини програми не в одному, а в декількох файлах. Кожний файл повинен включати вцілому одну чи декілька функцій. Імена цих файлів записуються в спеціальний файл – файл проекту, із якого IDE дізнається, які із текстових файлів треба об'єднувати у файл, що виконується (.EXE).

Всі необхідні для роботи з файлами проектів команди включені в меню ***Project***.

Для організації файлу проекту необхідно відкрити файл проекту. Для цього виконується команда *Project – Open Project…* IDE активізує спеціальне вікно Project у нижній частині екрану і відкриває вікно діалогу, що дозволяє завантажити потрібний файл проекту чи створити новий з заданим ім'ям.

Якщо створений новий файл проекту, вікно Project спершу буде пустим. Включення файлів у проект і їх видалення виконується або через команди *Project – Add item…* і *Project – Delete item*, або натисканням клавіш Ins і Del, у випадку якщо курсор розміщений у вікні Project. Під час добавляння файлів у проект відкривається вікно діалогу, що дозволяє вибрати потрібний файл.

Вікно Project спрощує перехід від одного файлу, що включений до проекту, до іншого під час їх редагування. Для цього висвітлювання переміщається на потрібний рядок вікна Project і натискається клавіша ENTER.

Під роботи з середовищем Borland C++ рекомендується використовувати проект, навіть якщо програма складається із одного файлу.

Відлагодження програми. В процесі відлагодження можна: 1)виконувати покрокове виконання програми. Після проходу кожного її рядка буде відбуватися призупинення, що дозволяє проаналізувати проміжні результати;

2)перевіряти значення і розташування (адресу) деякої змінної в ході виконання програми;

3)переглянути послідовність виклику функцій у програмі.

Існують два режими покрокового виконання програми:

1)трасування з заходом в тіло функції, під час зустрічі її виклику в тексті програми (F7);

2)покрокове виконання функції (як звичайної команди без заходу в тіло функції), під час зустрічі її виклику в тексті програми (F8).

Команда *Run – Trace into* (F7) запускає програму на відлагодження. Інтегроване середовище висвітлює рядок програми, що містить точку входу main(). Після цього натисканням клавіші F7 відбувається виконання коду, що відповідає одному рядку тексту програми. Якщо у рядку записане посилання на функцію, починається трасування по тексту тіла функції. За необхідністю виконання рядка функції за один крок, використовують клавішу F8 (команда *Run – Step over*).

Для прискорення процесу від лагодження використовується команда *Run – Go to cursor* (F4). Програма виконується до рядка, в якому в даний момент розташований текстовий курсор. Можна також задати режим виконання до точки зупинення (через опцію підменю *Debug – Toggle breakpoint* чи натисканням клавіші Ctrl+F8, При цьому рядок в точці зупинки підсвічується зазвичай червоним фоном. Зняти установлену точку зупинки можна повторним виконанням описаної команди, розмістивши курсор на підсвіченому рдку зупинки.

Для спостеріганням за змінами значень змінних в ході виконання програми використовується підменю *Debug – Watches\Add watch* чи Ctrl+F7. У вікні, що з’явиться Add Watch (виклик вікна Add Watch можна також отримати якщо натиснути клавішу Ins, попередньо зробивши активним вікно Watch) необхідно ввести ім’я змінної, значення якої треба переглянути і натиснути ENTER. Вказана змінна розміщується у вікні Watch, що створюється у нижній частині екрану, у в процесі відлагодження через це вікно можна спостерігати за змінами змінних, що розміщені в ньому. Видалити змінну із вікна Watch можна за допомогою клавіші Del, попередньо виділивши змінну підсвічуванням.

Використовуючи опцію меню *Evaluate/modify* чи Ctrl+F4, можна змінити значення змінної в процесі виконання відлагодження, щоб протестувати алгоритм з новим заданим значенням. Вікно цієї опції *Evaluate and Modify* можна використовувати і в якості калькулятора, якщо записати вираз зі змінними у рядку Expression і натиснути клавішу Evaluate для отримання результату у рядку Result.

ДОДАТОК Б – ПРИКЛАД ТИТУЛЬНОГО ЛИСТА ЗВІТУ ПО ЦИКЛУ ЛАБОРАТОРНИХ РОБІТ

Міністерство освіти і науки України
Чернігівський державний технологічний університет

Кафедра промислової електроніки

ЗВІТ

про виконання циклу лабораторних робіт
по курсу ”**Програмування та алгоритмічні мови** ”

Виконав: студент групи (*шифр групи*)
(П.І.Б. студента)

Перевірив: (*П.І.Б. викладача*)

Чернігів (*рік*)