Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

ЕЛЕМЕНТИ СТРУКТУРНОГО ПРОГРАМУВАННЯ

Методичні рекомендації
до практичних занять з курсу «Програмування»
для студентів спеціальностей 113 «Прикладна математика»,
142 «Енергетичне машинобудування», 143 «Атомна енергетика»,
174 «Автоматизація, комп'ютерно-інтегровані технології
та робототехніка»

Електронне видання

Харків – 2023

УДК 519.6: 519.7: 621.039: 681.3
Е 50

Рецензенти:

С. Л. Гефтер – кандидат фізико-математичних наук, доцент кафедри фундаментальної математики Харківського національного університету імені В. Н. Каразіна;
О. М. Цимбал – доктор технічних наук, професор кафедри комп'ютерно-інтегрованих технологій, автоматизації та мехатроніки Харківського національного університету радіоелектроніки.

Затверджено до розміщення в мережі Інтернет рішенням Науково-методичної ради Харківського національного університету імені В. Н. Каразіна (протокол № 9 від 16.06.2023 р.)

Е 50 **Елементи** структурного програмування : методичні рекомендації до практичних занять з курсу «Програмування» для студентів спеціальностей 113 «Прикладна математика», 142 «Енергетичне машинобудування», 143 «Атомна енергетика», 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» [Електронне видання] / уклад. В. І. Коробов, Ю. В. Ромашов, К. В. Стєпанова. – Харків : ХНУ імені В. Н. Каразіна, 2023. – (PDF 36 с.)

Методичні рекомендації підготовлені щодо вивчення теми «Елементи структурного програмування» відповідно до основного змісту навчальної дисципліни «Програмування». Обговорюються базові принципи щодо організації циклічного повторення виконання послідовностей інструкцій, використання підпрограм, створення програмних одиниць для багаторазового використання, виконання заданої кількості циклів послідовностей інструкцій.

Методичні рекомендації призначені для здобувачів вищої освіти різних курсів та рівнів за спеціальностями 113 «Прикладна математика», 142 «Енергетичне машинобудування», 143 «Атомна енергетика», а також 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка».

УДК 519.6: 519.7: 621.039: 681.3

ЗМІСТ

ВСТУП

Комп'ютеризація (digitalization) сьогодні є одним із ключових засобів забезпечення сталого розвитку суспільства та економіки, тому до неї приділяється багато уваги в розвинених країнах, у країнах ЄС [1] тощо. Комп'ютеризація передбачає розробку й упровадження відповідних програмних засобів, у тому числі для наукової та інжинірингової діяльності, що забезпечує розв'язування багатьох сучасних глобальних проблем [1]. Отже, вивчення програмування є важливим для навчання майже за кожною спеціальністю. Ці методичні вказівки призначені для здобувачів вищої освіти різних курсів та рівнів за спеціальностями 113 Прикладна математика, 142 Енергетичне машинобудування, а також 143 Атомна енергетика й 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка, які хоча й відносяться до різних галузей, але мають багато спільних інтересів щодо математичного забезпечення програмних засобів різного призначення, у тому числі для переходу до кліматично нейтральної енергетики, що є однією із важливих глобальних проблем сучасності та одним із пріоритетів діяльності ЄС [1] тощо.

Широке впровадження комп'ютерів та програмних продуктів в суспільство, економіку, наукові дослідження та індустрію призвело до необхідності супроводу розроблених програм, який передбачає виправлення помилок та внесення модифікацій з метою забезпечення коректного та більш зручного подальшого користування. Структурний підхід щодо програмування [2], в якому пропонується створювати програми у вигляді сукупності типових структур, було запропоновано у тому числі саме для забезпечення можливостей для більш зручного розуміння тексту програм, пошуку помилок та їх модифікації.

Метою цих методичних вказівок до практичних занять з курсу програмування є засвоєння теоретичного матеріалу щодо базових понять та концепцій структурного програмування, а також його використання при розробці програм для наукових та інженерних розрахунків.

Реалізація зазначеної мети буде здійснюватися шляхом поступового створення від простіших до більш складних програм для виконання наукових та інженерних розрахунків, в яких доцільно використовувати саме структурне програмування. Буде використано мову програмування FORTRAN 77, яка є зручною для первинного навчання програмуванню, оскільки на основі цієї мови програмування було створено велику кількість різноманітних сучасних мов програмування, та завдяки широкому використанню сьогодні удосконалених версій FORTRAN щодо розробки програм для наукових та інженерних розрахунків [3, 4]. Для виконання завдань рекомендується використовувати доступний для вільного користування компілятор проекту Open Watcom, який можна без обмежень завантажувати із мережі Internet [5, 6].

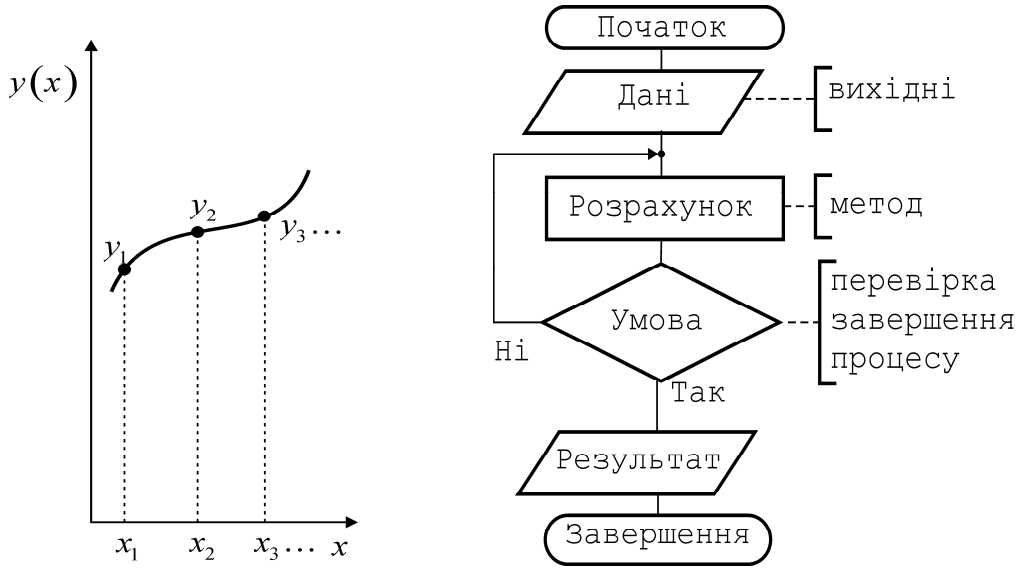
1 ЦИКЛІЧНИЙ ПОВТОР ВИКОНАННЯ ПОСЛІДОВНОСТЕЙ ІНСТРУКЦІЙ

Циклічні повторення послідовності інструкцій є одним із елементів структурного програмування, тобто такі повторення є необхідними для створення програм різного призначення.

1.1 Теоретичні відомості щодо циклічного виконання інструкцій

Розглянемо далі базові поняття щодо циклічного виконання інструкцій у програмах на прикладі типових обчислювальних методів, що використовуються для розв'язування задач прикладної математики при виконанні наукових та інженерних розрахунків різного призначення.

1.1.1 Якщо математичну задачу не можна розв'язати шляхом аналітичних перетворень, то можна скористатися обчислювальними методами. Для цього спочатку розв'язок задачі слід представити набором чисел, як, наприклад, функція представляється її значеннями в окремих точках (рис. 1.1а). Далі слід запропонувати обчислення, циклічне виконання яких приведе нас до чисел, що представляють розв'язок розглянутої задачі. Не завжди кількість циклічних повторень обчислень, необхідну для одержання розв'язку, можна визначити заздалегідь, тому завершення циклу обчислень слід здійснювати за деякою відповідним чином встановленою умовою. Наприкінці слід розробити програму за схемою, що показана на рис. 1.1б, та виконати необхідні обчислення.



а) б)

а) представлення шуканого розв'язку набором чисел;

б) схема алгоритму програми розрахунку

Рисунок 1.1 – Розв'язок математичної задачі обчислювальними методами

Зрозуміло, що реалізація програми, яка містить циклічні повторення послідовності інструкцій, як, наприклад, показано вище на рис. 1.1, вимагатиме зміни прийнятої за замовчанням послідовності виконання інструкцій за їхнім розташуванням в тексті програми.

1.1.2 Для зміни прийнятої за замовчанням послідовності виконання інструкцій за їхнім розташуванням в тексті програми з метою організації циклічного виконання заданої послідовності інструкцій, як, наприклад, на рис. 1.1, можна використовувати відповідні спеціальні інструкції безумовного та (або) умовного переходу, що визначають, яку саме інструкцію слід виконувати із поточного місця програми. Нагадаємо, що в мові програмування FORTRAN передбачені наступні інструкції безумовного та умовних переходів:

$$\text{GOTO label}, \tag{1.1}$$

$$\text{IF (ArExpr) label1, label2, label3}, \tag{1.2}$$

де label1, label2 та label3 – унікальні мітки, які заздалегідь визначені, щоб помітити відповідні інструкції програми; ArExpr являє собою деякий арифметичний вираз або ім'я деякої змінної з числовим типом.

Слід зазначити, що використання інструкції (1.2) умовного переходу передбачає введення як мінімум двох міток та призводить до значного збільшення кількості міток в програмах. Наявність великої кількості міток, зрозуміло, ускладнює читання програм взагалі, і виключення міток із програм є одним із принципів структурного програмування [2]. В той же час, в деяких випадках використання невеликої кількості міток може навпаки значно спростити розробку та читання програм, особливо наукового та інженерного призначення, тому технікою використання міток в програмах слід володіти.

1.1.3 Для зміни прийнятої за замовчанням послідовності виконання інструкцій за їхнім розташуванням в тексті програми з метою організації циклічного виконання заданої послідовності інструкцій, як, наприклад, на рис. 1.1, замість інструкцій безумовного та переходів (1.1) та (1.2) можна використовувати також наступні інструкції:

$$\text{IF (LogicExpr) Instruction}, \tag{1.3}$$

```
IF (cond) THEN
instr for if
ELSE
instr for else
END IF
```
(1.4)

де LogicExpr – логічний вираз; Instruction – інструкція, яка має бути виконана при задовільнені мови, що формується виразом LogicExpr ; cond –

змінна або вираз логічного типу, якими визначають необхідну умову; instr for if – інструкції, які мають бути виконані за умов, коли значення cond дорівнює .TRUE., а instr for else – інструкції, які мають виконуватися у протилежному випадку.

Використання інструкції (1.3) замість інструкції (1.2) може в деяких випадках помітно зменшити кількість міток в програмі, але найбільшого ефекту щодо спрощення розробки та читання програм можна досягти за рахунок використання інструкції (1.4), вигляд якої повністю узгоджений із концепцією структурного програмування. Інструкція (1.4) дозволяє забезпечувати виконування заданої послідовності інструкцій при виконанні заданої певної умови. Слід підкреслити, що використання конструкцій вигляду IF–THEN–ELSE є характерним саме для структурного програмування та здійснюється однаково в багатьох універсальних мовах програмування, таких як C, Python та багатьох інших.

1.1.4 Виконання наукових та інженерних розрахунків часто потребує обчислення тригонометричних функцій, експоненти, логарифмів, модуля та інших стандартних математичних функцій. В мові програмування FORTRAN, зрозуміло, передбачені можливості обчислення стандартних математичних функцій. Наприклад, обчислення значення $\mathrm{tg}(x)$ в FORTRAN здійснюється так:

$$\mathrm{TAN(X)}, \tag{1.5}$$

де X – змінна, або числова константа, яка визначає аргумент x, для якого слід визначити $\mathrm{tg}(x)$.

Окрім можливості обчислення тангенсу (1.5) в мові програмування FORTRAN передбачені також інші можливості обчислення стандартних математичних функцій [3, 4], деякі із яких наведені в табл. 1.1.

Таблиця 1.1 – Обчислення математичних функцій в FORTRAN

Математична функція	Запис на мові програмування
$\cos(x)$	COS(X)
$\sin(x)$	SIN(X)
$\arccos(x)$	ACOS(X)
e^x	EXP(X)
$\ln(x)$	LOG(X)
$\lvert x \rvert$	ABS(X)
$\lg(x)$	LOG10(X)
$\min(x_1, x_2)$	MIN(X1,X2)

1.2 Завдання щодо програм із циклічним виконанням інструкцій

Реалізацію розглянутих загальних рекомендацій щодо використання циклічного повторення послідовностей інструкцій в програмах далі покажемо на прикладі розв'язування математичної задачі про пошук кореня рівняння у заданому інтервалі:

$$x: \quad f(x)=0, x\in[a,b], \tag{1.6}$$

де $f(x)$ – функція довільного вигляду; a, b – границі інтервалу, в якому визначається корінь рівняння.

1.2.1 Розв'язок задачі (1.6) є числом, тому будуємо обчислювальну схему для безпосереднього розв'язування цієї задачі – визначення значення кореня. Для цього скористаємося методом половинного ділення інтервалу, який засновано на відомому факті математичного аналізу, що при існуванні єдиного кореня рівняння (1.6) в інтервалі $[a,b]$ виконується умова $f(a)f(b)<0$. В цих умовах маємо, що середина $x_1=(a+b)/2$ інтервалу $[a,b]$ має бути більш близькою до шуканого кореня, ніж один із країв цього інтервалу. До якого саме краю є ближчим корінь рівняння, можна встановити перевіркою умови

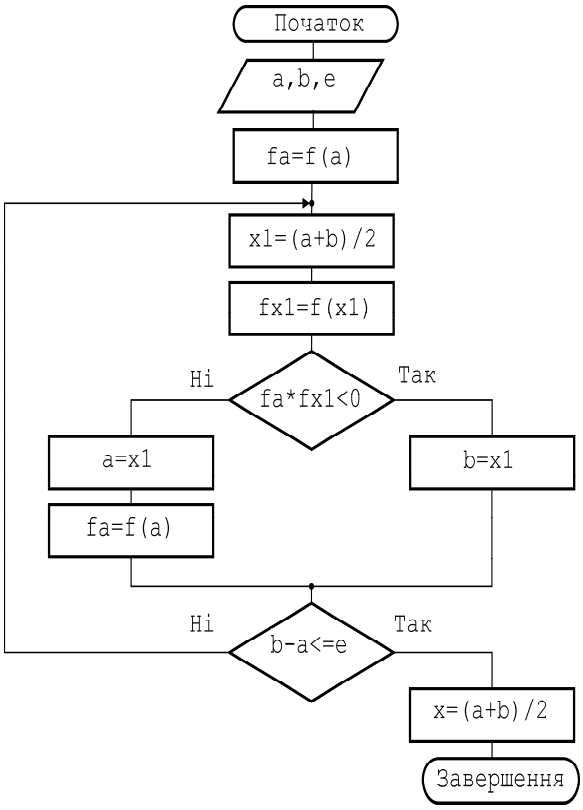


Рисунок 1.2 – Схема алгоритму методу половинного ділення

$$f(a)f(x_1)<0. \tag{1.7}$$

Якщо умова (1.7) виконується, то край a є ближчим до шуканого кореня, який далі можна визначати на меншому інтервалі $[a,b]$, де $b=x_1$; при невиконанні цієї умови край b інтервалу є ближчим до шуканого кореня, який можна далі визначати на меншому інтервалі $[a,b]$, де $a=x_1$. Далі слід знову визначити середину $x_1=(a+b)/2$ нового інтервалу $[a,b]$, перевірити умову (1.7) і відповідним чином зменшити довжину $\varepsilon=b-a$ інтервалу $[a,b]$, яка є похибкою визначення кореня. Довжину $\varepsilon=b-a$ інтервалу слід зменшувати, поки вона не стане достатньо малою, щоб представляти числове значення шуканого кореня. Таким чином, шляхом циклічного ділення інтервалу на дві частини наближаємося до кореня рівняння; схема алгоритму визначення кореня рівняння (1.6) представлена на рис. 1.2.

1.2.2 Розглядатимемо далі можливі варіанти програм розв'язання рівняння (1.6), яке реалізується методом половинного ділення інтервалу, як представлено на рис. 1.2. В якості функції $f(x)$ використовуємо:

$$f(x) = \mathrm{tg}(x) - x\,. \quad (1.8)$$

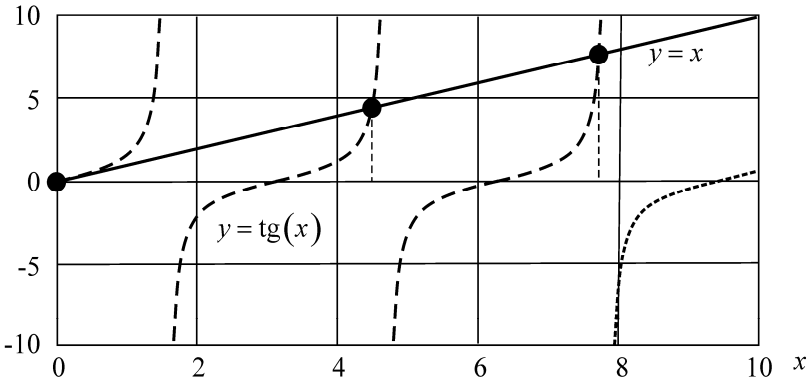


Рисунок 1.3 – Геометричне розв'язування рівняння

Розв'язки рівняння (1.6), що відповідає функції (1.8), можна наочно уявляти як ординати точок перетину графіків функцій $y = x$ та $y = \mathrm{tg}(x)$, як показано на рис. 1.3; бачимо (рис. 1.3), що рівняння (1.6) для функції (1.8) має нескінченну кількість коренів, тому далі розглянемо визначення кореня рівняння (1.6) для функції (1.8) в інтервалі, який має границі:

$$a = 3, \quad b = 5\,. \quad (1.9)$$

1.3 Приклади програм із циклічним виконанням інструкцій

Розглянемо далі приклади програм, в яких циклічне виконання послідовності інструкцій здійснюється для розв'язування рівняння (1.6) з вихідними даними (1.8), (1.9) методом половинного ділення.

1.3.1 Текст програми, написаний на мові програмування FORTRAN, в якій здійснюється розв'язування рівняння (1.6) для функції (1.8) в інтервалі, що визначається границями (1.9), має містити дві інструкції керування та може мати наступний вигляд:

Лістинг 1.1 – Текст програми визначення кореня рівняння

```
      PROGRAM ROOT1
      REAL A,B,X1,X,FA,FX1,E
      A=3.0
      B=5.0
      E=1.0E-6
      FA=TAN(A)-A
100   X1=(A+B)/2
      FX1=TAN(X1)-X1
      IF ((FA*FX1).LT.0) THEN
      B=X1
      ELSE
      A=X1
```

```
      FA=TAN(A)-A
      END IF
      IF ((B-A).GT.E) GOTO 100
      X=(A+B)/2
      PRINT*,'X=',X
      PRINT*,'TAN(X)=',TAN(X)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Представлена програма (ліст. 1.1) дозволяє одержати розв'язок рівняння (1.6) для функції (1.8) та границь інтервалу (1.9) із високою точністю. В програмі не передбачається введення вихідних даних, тобто вона дозволяє одержувати один із коренів рівняння (1.6) для вихідних даних (1.8) та (1.9) з похибкою $\varepsilon = 10^{-6}$, але в такій програмі нескладно передбачити введення вихідних даних таким чином, щоб можна було визначати будь-який із коренів рівняння (1.6) для функції (1.8) із заданою точністю, що можна запропонувати для самостійного виконання. Циклічне виконання інструкцій в програмі на ліст. 1.1 реалізовано шляхом використання мітки та інструкції умовного переходу.

1.3.2 Завершення циклічних розрахунків здійснюється шляхом перевірки відповідної умови, але може статися так, що ця умова не буде виконана, наприклад, через наближений характер обчислень, і в цьому випадку програма буде нескінченно здійснювати розрахунки. Для уникнення подібних випадків при виконанні циклічних розрахунків передбачають обчислення кількості виконаних циклів, щоб мати можливість обмежити кількість циклів обчислень. В мові програмування FORTRAN передбачені змінні цілого типу INTEGER, які дозволяють, наприклад, підраховувати кількість виконаних циклів обчислень, що мають бути представлені цілими числами. Приклад програми, що розв'язує рівняння (1.6) для функції (1.8) в інтервалі з границями (1.9), може мати такий вигляд:

Лістинг 1.2 – Текст програми визначення кореня рівняння із підрахуванням кількості циклів обчислень

```
      PROGRAM ROOT2
      REAL A,B,X1,X,FA,FX1,E
      INTEGER I,N
      A=3.0
      B=5.0
      N=100
      E=1.0E-6
      FA=TAN(A)-A
      I=0
100   X1=(A+B)/2
      FX1=TAN(X1)-X1
```

```
I=I+1
IF ((FA*FX1).LT.0) THEN
B=X1
ELSE
A=X1
FA=TAN(A)-A
END IF
IF (((B-A).GT.E).AND(I.LE.N)) GOTO 100
X=(A+B)/2
PRINT*,'X=',X
PRINT*,'TAN(X)=',TAN(X)
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
```

В програмі, що наведена на ліст. 1.2, для визначення кількості здійснених циклів розрахунків, а також для визначення максимально можливої кількості циклічних обчислень для наближеного розв'язування рівняння (1.6) передбачені дві змінні цілого типу. Перед початком циклічних розрахунків змінній цілого типу – лічильнику кількості циклів слід присвоїти нульове значення, а при закінченні кожного циклу обчислень – збільшувати значення цієї змінної на одиницю. В умові, що визначає закінчення обчислень, окрім перевірки поточної довжини інтервалу визначення кореня рівняння, слід врахувати також умову, щоб поточна кількість виконаних циклів розрахунків не перевищувала заданого значення. Максимальна можлива величина кількості циклів розрахунку має бути достатньо великою величиною, яка має перевищувати кількість циклів, що потрібні для розв'язування рівняння із заданою точністю. При вдало побудованому алгоритмі розв'язування задачі кількість циклів розрахунків, необхідних для одержання розв'язку задачі із потрібною точністю, не буде дуже великою.

1.4 Контрольні запитання та завдання

Наявність знань, що підтверджують засвоєння матеріалу заняття, можна перевірити спроможністю надати відповіді та виконати наступні питання та завдання:

1. Як здійснюється використання обчислювальних методів для розв'язування математичних задач?
2. Навести схему алгоритму програми, що здійснює використання обчислювального методу для розв'язування математичної задачі.
3. Які інструкції мови програмування FORTRAN можна використовувати для організації циклічного виконання послідовностей інструкцій?
4. Які інструкції мови програмування FORTRAN краще використовувати для організації циклічного виконання послідовностей інструкцій?

5. Як наближено визначити корінь рівняння методом половинного ділення?

6. Навести схему алгоритму програми визначення кореня рівняння методом половинного ділення.

7. Розробити програму наближеного розв'язування рівняння (1.6) методом половинного ділення для вихідних даних:

$$f(x) = x^3 - 6x^2 + 12x - 8,\ \ a = 1,\ \ b = 3{,}5\,. \qquad (1.10)$$

2 ВИКОРИСТАННЯ ПІДПРОГРАМ

В деяких програмах необхідно багато у різних місцях виконувати однакові послідовності деяких заданих інструкцій при різних значеннях деяких із змінних, і текст такої програми буде дуже громіздким, якщо в ній у великій кількості різних місць будуть повторюватися однакові послідовності деяких заданих інструкцій.

2.1 Теоретичні відомості щодо використання підпрограм

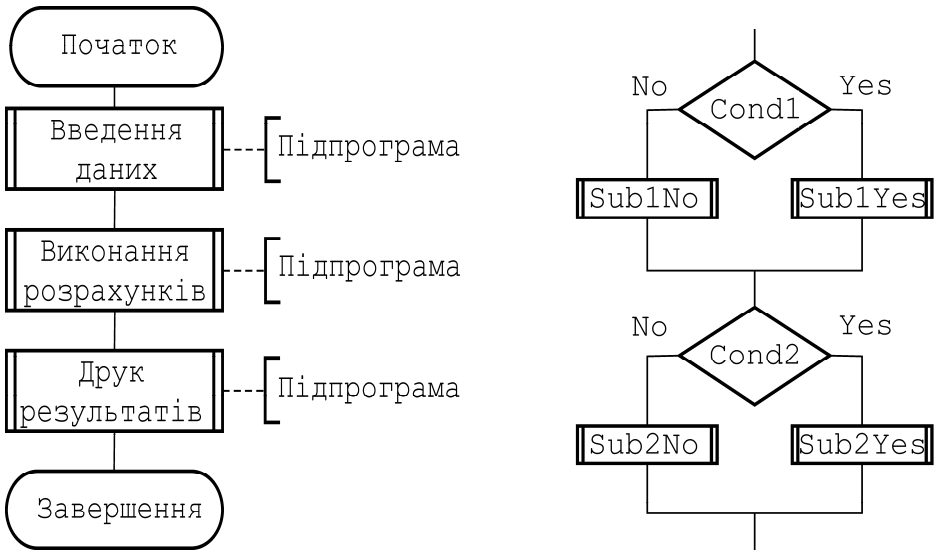
В мовах програмування зазвичай передбачається можливість одноразового окремого завдання послідовностей інструкцій для подальшого багаторазового їхнього використання в програмах [3, 4], що є одним із елементів структурного програмування, яке забезпечує необхідні властивості створюваним програмам.

2.1.1 Підпрограма є одним із способів організації послідовності інструкцій, що використовується в структурному програмуванні, і являє собою незалежну послідовність деяких заданих інструкцій, виконання якої можна здійснювати із будь-якого місця програми із подальшим поверненням до наступної інструкції цієї програми. Кажуть, що підпрограма «викликається», тобто програма передає керування відповідній підпрограмі. Після завершення свого виконання викликана підпрограма повертає керування назад до програми, що її викликала, таким чином, що керування буде передано інструкції програми, яка розташована слідом за інструкцією, що викликала підпрограму.

Використання підпрограм суттєво спрощує розробку та читання текстів програм, оскільки дозволяє раціоналізувати будову програм шляхом їхнього структурування. Так, програма може містити три підпрограми, що виконуються послідовно: підпрограму введення даних, підпрограму розрахунків, підпрограму друку результатів на екрані монітора (рис. 2.1а). Підпрограми також можна використовувати для спрощення структурної інструкції IF за рахунок більш компактного її представлення за рахунок

використання коротких викликів підпрограм замість довгих послідовностей інструкцій, що мають виконуватися при виконанні тих чи інших умов (рис. 2.1б). Звичайно, що питання про доцільність використання підпрограм має розв'язуватися в кожному конкретному випадку.

В деяких мовах програмування, наприклад в FORTRAN та PASCAL, передбачені два різновиди підпрограм: підпрограми-функції та підпрограми-процедури, а в деяких мовах програмування, наприклад в С, такого розділення підпрограм не передбачається, й усі підпрограми є функціями. В деяких мовах програмування немає жодних інших конструкцій, окрім підпрограм-процедур та підпрограм-функцій, і будь-яка програма представляється у вигляді послідовностей підпрограм-процедур та підпрограм-функцій. В будь-якому випадку потрібно розуміти різницю між підпрограмами-процедурами та підпрограмами-функціями для використання у подальшому різних мов програмування. Підпрограми є окремими програмними одиницями та визначаються окремо від програми; в мові програмування FORTRAN підпрограми необхідно визначати після інструкції END завершення програми.



а) б)

а) виділення етапів програми; б) структуризація програми

Рисунок 2.1 – Приклади використання підпрограм

2.1.2 Поняття про підпрограму-функцію в програмуванні суттєво пов'язане із поняттям функції в математиці, яке формулюють наступним чином: нехай маємо множину, яка складається із елементів x, а також ножину Y, яка складається із елементів y, і нехай якимось способом f елементу $x \in X$ ставиться у відповідність елемент $y \in Y$, тоді відповідність $x \to y$, або $y = f(x)$ називають функцією із областю визначення X та областю значень Y. В програмуванні під поняттям підпрограма-функція розуміють підпрограму, яка має певний результат у вигляді значення даних певного типу, що визначається певним чином за набором вхідних пара-

метрів, що у загальному випадку являють собою множину значень даних певних типів. Тобто підпрограма-функція є аналогом математичної функції таким чином, що результат підпрограми-функції відповідає значенню функції в математичному сенсі, а множина вхідних даних підпрограми-функції – аргументу функції в математичному сенсі.

При виклику підпрограми-функції керування передається відповідній підпрограмі, і в результаті по закінченні інструкцій цієї підпрограми до програми передається результат – значення функції. Передбачається, що підпрограми-функції визначаються окремо та можуть використовуватися у будь-якому місці програмі. Природно використовувати підпрограми-функції, коли в декількох місцях програми потрібно виконувати однакові послідовності інструкцій при різних значеннях деякого заданого набору змінних програми; в цьому випадку в якості аргументу функції доцільно передбачити відповідний заданий набір змінних, а саму функцію виконати у вигляді відповідної послідовності інструкцій.

Підпрограма-функція в мові програмування FORTRAN визначається за допомогою інструкції FUNCTION, після якої розташовують інструкції підпрограми-функції, які мають завершитися інструкцією END, що взагалі можна представити наступним чином [3, 4]:

```
type FUNCTION name(arguments)
...
name = ...
END
```

В описі підпрограми-функції слід визначити тип даних значення функції, що здійснюється за допомогою безпосереднього вказування типу type; слід також визначитися з іменем функції, яке має міститися в name, а також із вхідними параметрами (аргументами), що позначені як arguments та мають вказуватися через кому в дужках одразу ж після імені функції. Серед інструкцій підпрограми функції обов’язково має міститися інструкція присвоювання змінної, ім’я якої збігається із іменем функції, що необхідно для визначення значення функції. Типи вхідних параметрів підпрограми-функції мають бути визначені за допомогою відповідних інструкцій, як це робиться в програмах; підпрограма-функція може мати також «локальні» змінні, тип яких визначається за допомогою інструкцій визначення, як у програмах. Якщо в програмі використовуються підпрограми-функції, то їхні імена мають бути визначені разом із змінними програми.

2.1.3 Поняття про підпрограму-процедуру зводиться до простої послідовності інструкцій, які мають бути виконані при зверненні з програми; після виконання інструкцій підпрограми-процедури керування передається інструкції програми, яка розташована одразу ж за інструкцією, що викликала цю підпрограму-процедуру.

Підпрограми-процедури в мові програмування FORTRAN [3, 4] визначають за допомогою інструкції SUBROUTINE, після якої розташовують необхідні інструкції, та завершують опис підпрограми-процедури інструкцією END, що взагалі має наступний вигляд:

```
SUBROUTINE name(arguments)
...
END
```

В описі підпрограми-процедури слід визначити її ім'я name, за допомогою якого програма буде викликати цю підпрограму-процедуру, а також вхідні параметри arguments, що мають вказуватися через кому в дужках одразу ж після імені. Серед інструкцій підпрограми-процедури обов'язково мають міститися визначення типів вхідних параметрів, а також «локальних» змінних, тип яких має бути визначений відповідним чином, як у програмах. Виклик підпрограм-процедур [3, 4] здійснюється за допомогою інструкції CALL, після якої вказується ім'я підпрограми-процедури та за необхідністю в дужках наводиться список її вхідних параметрів.

2.2 Приклади використання підпрограм

Прикладом доцільності використання підпрограми-функції є розв'язування рівняння (1.6) методом половинного ділення, в якому необхідно у двох місцях програми визначати значення функції $f(x)$ для різних значень аргументу x (рис. 2.2).

2.2.1 Правило обчислення значення функції $f(x)$ в програмі, що здійснює розв'язування рівняння (1.6), доцільно сформулювати окремо у вигляді підпрограми-функції та використовувати цю підпрограму в необхідних місцях програми.

Приклад програми, в якій для розв'язування рівняння (1.6) методом половинного ділення передбачено використовувати підпрограму-функцію, є наступним:

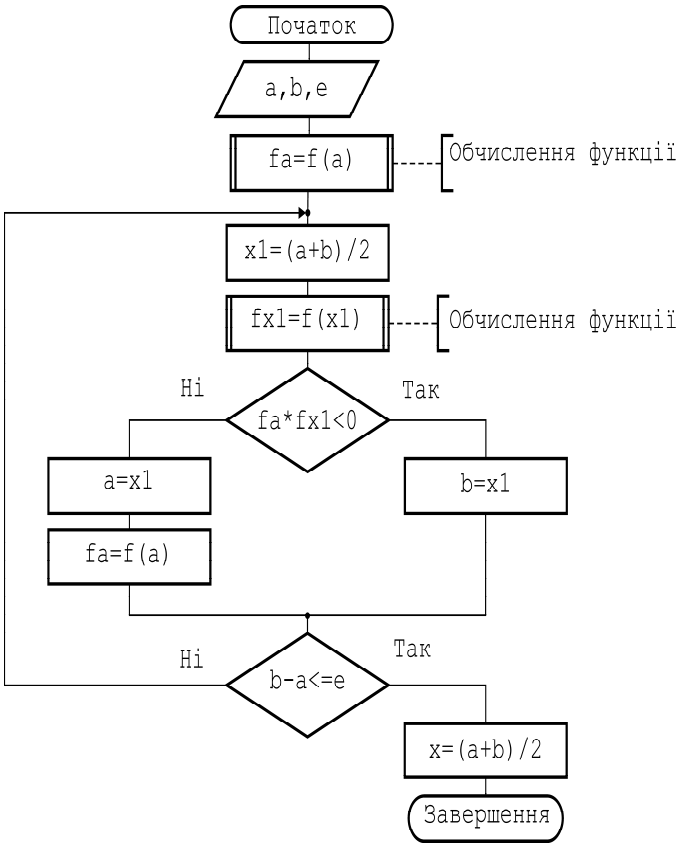


Рисунок 2.2 – Використання підпрограми-функції в алгоритмі методу половинного ділення

Лістинг 2.1 – Визначення кореня рівняння із використанням підпрограми-функції

```
      PROGRAM ROOT3
      REAL A,B,X1,X,FA,FX1,E,F
      A=3.0
      B=5.0
      E=1.0E-6
      FA=F(A)
100   X1=(A+B)/2
      X1=F(X1)
      IF ((FA*FX1).LT.0) THEN
      B=X1
      ELSE
      A=X1
      FA=FX1
      END IF
      IF ((B-A).GT.E) GOTO 100
      X=(A+B)/2
      PRINT*,'X=',X
      PRINT*,'TAN(X)=',TAN(X)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END

      REAL FUNCTION F(X)
      REAL X
      F=TAN(X)-X
      END
```

Виклик підпрограми-функції в програмі (ліст. 2.1) здійснюється шляхом звернення до її імені із переліченням в дужках через коми списку вхідних параметрів у вигляді констант та (або) імен змінних; результат підпрограми-функції може бути збережений у змінній, а також використаний у виразах різного типу.

2.2.2 У випадку, коли в програмі потрібно визначати корені рівняння для різних інтервалів, доцільно оформити визначення кореня рівняння (1.6) у вигляді окремої підпрограми-функції, вхідними параметрами якої є границі інтервалу пошуку кореня (два параметри), та допустима похибка (один параметр), як показано, наприклад, на ліст. 2.2. У наведеному прикладі границі інтервалу пошуку кореня є константами, але вони можуть бути також змінними, як і допустима похибка.

Лістинг 2.2 – Визначення кореня рівняння, що оформлене у вигляді окремої підпрограми-функції

```
      PROGRAM ROOT4
      REAL E,X1,X2,ROOT
```

```
      E=1.0E-6
      X1=ROOT(3.0,5.0,E)
      X2=ROOT(7.0,8.0,E)
      PRINT*,'X1=',X1,';   ','TAN(X1)=',TAN(X1)
      PRINT*,'X2=',X2,';   ','TAN(X2)=',TAN(X2)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END

      REAL FUNCTION ROOT(A,B,E)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT=(AA+BB)/2
      END

      REAL FUNCTION F(X)
      REAL X
      F=TAN(X)-X
      END
```

2.2.3 Прикладом доцільності використання підпрограми-процедури є розв'язування рівняння (1.6) методом половинного ділення у випадку, коли необхідно одразу ж одержати два корені для двох різних заданих інтервалів. В цьому випадку доцільно передбачити підпрограму із вхідними параметрами, що визначають границі двох інтервалів для визначення коренів (чотири параметри) та похибку визначення коренів (один параметр), та із результуючими параметрами, що визначатимуть знайдені корені (два параметри). Зрозуміло, що в цій програмі також доцільно використовувати підпрограму-функцію визначення кореня рівняння (1.6) у заданому інтервалі із заданою точністю та підпрограму-функцію для обчислення значення функції $f(x)$, відповідного заданому її аргументу x, що використовувались раніше в програмі, наведеній вище на ліст. 2.2. Приклад використання підпрограми-процедури щодо визначення коренів рівняння (1.6) одночасно в двох різних інтервалах наведений на ліст. 2.3.

Лістинг 2.3 – Визначення двох сусідніх коренів рівняння з використанням спеціальної підпрограми-процедури

```
      PROGRAM ROOT5
      REAL X1,X2
      CALL ROOTS(4.0,5.0,7.0,8.0,1.0E-6,X1,X2)
      PRINT*,'X1=',X1,';   ','TAN(X1)=',TAN(X1)
      PRINT*,'X2=',X2,';   ','TAN(X2)=',TAN(X2)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END

      SUBROUTINE ROOTS(A1,B1,A2,B2,E,X1,X2)
      REAL A1,B1,A2,B2,X1,X2,E,ROOT
      X1=ROOT(A1,B1,E)
      X2=ROOT(A2,B2,E)
      END

      REAL FUNCTION ROOT(A,B,E)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT=(AA+BB)/2
      END

      REAL FUNCTION F(X)
      REAL X
      F=TAN(X)-X
      END
```

2.3 Контрольні запитання та завдання

Наявність знань, що підтверджують засвоєння матеріалу заняття, можна перевірити спроможністю надати відповіді та виконати наступні питання та завдання:

1. Що таке підпрограма, для яких цілей вона потрібна?
2. Які існують різновиди підпрограм та в чому їхня різниця?

3. Що таке підпрограма-функція, як вона оформлюється в мові програмування FORTRAN та як її слід використовувати в програмах?

4. Що таке підпрограма-процедура, як вона оформлюється в мові програмування FORTRAN та як її використовувати в програмах?

5. Навести схему алгоритму методу половинного ділення щодо визначення кореня рівняння (1.6) у заданому інтервалі та показати, в яких місцях доцільно використовувати підпрограму-функцію.

8. Розробити програму наближеного розв'язування рівняння (1.6) методом половинного ділення для вихідних даних (1.10) та передбачити підпрограму-функцію для обчислення функції $f(x)$, яка визначає розв'язуване рівняння.

6. Пояснити, чому метод половинного ділення щодо визначення кореня рівняння (1.6) у заданому інтервалі доцільно оформляти у вигляді окремої підпрограми-функції.

7. Розробити програму наближеного розв'язування рівняння (1.6) методом половинного ділення для вихідних даних (1.10) та передбачити підпрограму-функцію для обчислення функції $f(x)$, яка визначає розв'язуване рівняння, а також підпрограму-функцію для визначення кореня у заданому інтервалі.

8. Пояснити доцільність використання підпрограми-процедури щодо розв'язування рівняння (1.6) методом половинного ділення у випадку, коли необхідно одразу ж одержати два корені для двох різних заданих інтервалів.

9. Пояснити доцільність використання підпрограми-процедури щодо розв'язування рівняння (1.6) методом половинного ділення у випадку, коли необхідно одразу ж одержати два корені для двох різних функцій $f(x)$ у двох відповідних заданих інтервалах.

10. Розробити програму наближеного розв'язування рівняння (1.6) методом половинного ділення для вихідних даних (1.8), (1.9) та (1.10). Передбачити підпрограму-процедуру для одночасного обчислення двох коренів та окремі підпрограми-функції для обчислення функцій $f(x)$, яка визначає розв'язуване рівняння, та для визначення кореня у заданому інтервалі.

3 СТВОРЕННЯ ПРОГРАМНИХ ОДИНИЦЬ ДЛЯ БАГАТОРАЗОВОГО ВИКОРИСТАННЯ

При створенні великих програм окремі послідовності інструкцій зручно групувати в окремих програмних компонентах та складати великі програми із таких програмних компонент [3, 4]. Деякі із програмних компонент природно можуть бути використані в різних програмних проектах, як, наприклад, ті, що здійснюють стандартні математичні обчислення –

визначення коренів рівнянь, визначених інтегралів тощо. Використання готових програмних компонент, що виконують стандартні завдання, суттєво прискорює створення великих програмних систем, тому передбачається в усіх розвинених мовах програмування і є елементом структурного програмування.

3.1 Теоретичні відомості щодо створення програмних одиниць для багаторазового використання

Програмні компоненти, що призначені для багаторазового використання, мають бути підготовлені за відповідними правилами та належним чином забезпечені описами.

3.1.1 В деяких випадках буває зручним передавати підпрограму (функцію або процедуру) як аргумент до іншої підпрограми [3, 4]. Можливість розглядати підпрограму як аргумент іншої підпрограми надає надвеликих можливостей щодо створення бібліотек підпрограм різного призначення, які можуть бути використані в різних програмах.

Для забезпечення передачі підпрограми в якості аргументу до іншої підпрограми в списку вхідних параметрів (аргументів) цій іншій підпрограми слід передбачити відповідний формальний параметр. При виконанні підпрограми, що містить підпрограму в якості аргументу, виклик цієї підпрограми-аргументу здійснюється як у звичайних програмах. При цьому у програмній компоненті, що викликає підпрограму, яка містить інші підпрограми в якості аргументів, усі підпрограми, що використовуються в якості аргументів, мають бути позначені інструкцією EXTERNAL, щоб відрізняти імена підпрограм-аргументів від імен змінних.

3.1.2 Великі програми зручно представляти як сукупність відносно незалежних програмних компонент, що дозволяє принаймні розділити роботу програмістів шляхом зосередження на певних програмних компонентах та є притаманним сучасним технологіям та передбачено в усіх сучасних мовах програмування.

При розділенні програми на програмні компоненти слід визначитися, яка саме із компонент отримуватиме керування на початку виконання програми; така програмна компонента називається головною програмою. В мові програмування FORTRAN головна програма визначається інструкціями, що містяться між інструкціями PROGRAM та END, а усі інші програмні компоненти – підпрограми – отримують керування від основної програми. В той же час, нескладно уявити, що тексти великих програм в одному файлі будуть виглядати дуже громіздкими, і деякі із програмних компонент зручно розташовувати в окремих файлах. Зручність такого розділення пов’язана із тим, що підпрограма-функція визначення кореня

методом половинного ділення може бути розроблена окремо і використовуватися в різних програмах, і було б краще, щоб розроблений та перевірений код цієї підпрограми був огороджений від ненавмисних помилкових виправлень, при маніпуляціях в тексті основної програми.

Забезпечення можливостей створення програм, що складаються із декількох файлів, при використанні мови програмування FORTRAN здійснюється засобами інтегрованого середовища розробки програм IDE (Integrated development Environment), які розробляються окремо від компіляторів та надають можливостей щодо розроблення програм. В IDE Open WATCOM передбачається поняття про проект та джерела (sources). Проект має мати ім'я; уся інформація про проект міститься в файлі проекту, що має розширення WPJ, та в низці інших файлів (з різними розширеннями), які автоматично створюються IDE таким чином, що імена усіх файлів проекту збігаються із іменем проекту (але ж мають різні розширення). Джерела (sources) містять дані про послідовності інструкцій програмних компонентів, підготовлені у вигляді окремого файлу, зміст якого має відповідати правилам мов програмування, а розширення має відповідати мові програмування; якщо джерело підготовлено на мові програмування FORTRAN 77, то його файл має мати розширення FOR. Імена файлів-джерел не обов'язково мають збігатися з іменем проекту або містити його складові та позначки, хоча іноді це дійсно зручно.

3.1.3 Багато підпрограм можуть бути використані в різних проектах, наприклад, для виконання певних стандартних обчислень. Деякі із таких підпрограм самі можуть використовувати інші підпрограми, виконані в окремих файлах, і звичайно, що при приєднанні такої складної підпрограми до проекту слід приєднувати також усі інші підпрограми, які використовує ця приєднувана підпрограма, навіть якщо ці підпрограми непотрібні в проекті безпосередньо (рис. 3.1). Це створює певні незручності, для виправлення яких запропоновано технологію використання статичних бібліотек підпрограм. Статична бібліотека – це файл-джерело, записаний у спеціальному форматі та представлений у вигляді файлу із розширенням LIB, в якому містяться підпрограми, які можна використовувати в проекті, а також «внутрішні» підпрограми, які не будуть використовуватися у проекті безпосередньо (рис. 3.1), але є необхідними для роботи тих підпрограм, що використовуватимуться в проекті. Звичайно, що приєднати до проекту один файл-LIB бібліотеки підпрограм значно простіше, ніж приєднувати до проекту окремо усі файли, необхідні для роботи відповідної бібліотеки підпрограм.

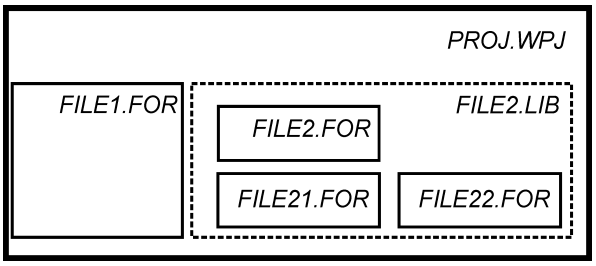


Рисунок 3.1 – Стандартна бібліотека та її використанн

3.1.4 Бібліотека підпрограм є проектом, при створенні якого засобами IDE у відповідь на питання про ціль (target) проекту в якості мети слід призначити статичну бібліотеку. Далі слід наповнити проект статичної бібліотеки необхідними джерелами – файлами із текстами необхідних підпрограм та здійснити компіляцію проекту, в якій буде створено файл із іменем проекту та розширенням LIB. Саме цей файл можна приєднувати до інших проектів, щоб використовувати присутні в ньому підпрограми. Слід зазначити, що зміст файлу-LIB бібліотеки підпрограм не можна змінювати і не можна з цього файлу одержати інформацію про підпрограми, які в ньому містяться. Для цього потрібно мати доступ до файлів-джерел, написаних на мові програмування, що були приєднані до проекту статичної бібліотеки при її компіляції. Тобто якщо в файлі-LIB бібліотеки підпрограм було виявлено помилку, наприклад, використання певної підпрограми приводить до помилкового результату, то необхідно внести зміни у відповідні файли-джерела, що написані мовою програмування та містяться в проекті цієї статичної бібліотеки, та заново компілювати бібліотеку. Ретельно перевірені статичні бібліотеки часто надають виключно у вигляді файлу-LIB, що не дозволить користувачу помилково змінити його, але файл-LIB бібліотеки підпрограм обов’язково має супроводжуватися ретельним описом усіх передбачених підпрограм, які в ньому реалізуються, щоб користувачі-програмісти мали можливість використовувати ці підпрограми в своїх проектах.

Бібліотеки підпрограм, представлені у вигляді файлів із розширенням LIB, називають статичними бібліотеками, оскільки вони цілком додаються до файлу програми-проекту, яка їх використовує. Бібліотеки підпрограм, тобто файли-LIB, додаються до проекту в якості джерела (source) разом із джерелами інших типів – файлів, написаних мовою програмування, наприклад файлів-FOR. Підпрограми, що містяться у бібліотеці, яка додана до проекту в якості джерела, використовуються в програмних компонентах проекту як звичайні підпрограми.

3.2 Типові завдання щодо розробки та застосування програмних одиниць для багаторазового використання

Прикладом завдання, для якого природно передбачати окремі програмні компоненти, що придатні для багаторазового використання, є задача (1.6) про визначення кореня рівняння у заданому інтервалі, наприклад, методом половинного ділення. Дійсно, уявімо собі, що в програмі на проміжному етапі великих обчислень для забезпечення розрахунків необхідно розв’язати задачу (1.6) про визначення кореня різних рівнянь у відповідних заданих інтервалах:

$$x: \quad f_1(x) = 0, x \in [a_1, b_1], \tag{3.1}$$

$$x: \quad f_2(x) = 0, x \in [a_2, b_2], \tag{3.2}$$

де $f_1(x)$ та $f_2(x)$ – задані функції; a_1, b_1 та a_2, b_2 – задані границі інтервалів, що містять корені відповідних рівнянь.

У випадку, коли функції $f_1(x)$ та $f_2(x)$, що визначають рівняння (3.1) та (3.2), збігаються, визначення різних коренів рівняння можна здійснити однією функцією, що реалізує, наприклад, метод половинного ділення, в якій передбачити границі інтервалу в якості вхідних параметрів – аргументів. У випадку, коли функції $f_1(x)$ та $f_2(x)$, слід створювати окремі функції для розв'язування рівнянь (2.1) та (2.2), але такі функції будуть відрізняться виключно викликами відповідної функції у відповідних місцях. Якщо в функції, що здійснює розв'язування задачі (1.6), окрім границь інтервалу $[a, b]$, який містить корінь, також передбачити використання функції $f(x)$ в якості вхідного параметра, то одна така функція може бути використана для розв'язування різних рівнянь, у тому числі (3.1) та (3.2), і це буде значно економити обсяг програмного коду в програмах, які потребують розв'язування багатої кількості різних задач (1.6).

Розроблену окремо відносно незалежну універсальну підпрограму-функцію, в якій здійснюється визначення коренів рівняння методом половинного ділення, зручно розташовувати в окремому файлі та використовувати цей файл як одне із джерел (source) програмного проекту (project). Зрозуміло, що такий підхід дозволить значно спростити текст основної програми та дозволить уникнути випадкового псування вже відпрацьованого програмного коду. Більш того, такий підхід дозволяє окремо створювати бібліотеки підпрограм, призначених для розв'язування типової задачі (1.6), оскільки для цього можна використовувати не тільки метод половинного ділення, а багато інших обчислювальних методів.

Хоча використання окремих файлів, що визначають підпрограми, в якості джерел проекту є досить зручним, але зміни в таких підпрограмах потребуватимуть повторної компіляції всього проекту. Використання статичних бібліотек підпрограм надає певні переваги, оскільки дозволяє у випадку зміни коду підпрограм такої бібліотеки просто замінити файл бібліотеки та не потребує повторної компіляції всього проекту.

3.3 Приклади розробки та застосування програмних одиниць для багаторазового використання

Покажемо використання арифметичної та логічної інструкції IF щодо зміни порядку виконання інструкцій відповідно сформульованому завданню.

3.3.1 Розглянемо спочатку розробку та застосування універсальної підпрограми-функції для визначення кореня рівняння в заданому інтервалі методом половинного ділення. Зрозуміло, що така підпрограма-функція має містити ім'я функції, яка визначає розв'язуване рівняння, в якості одного із своїх вхідних параметрів. Далі покажемо використання розробленої універсальної підпрограми-функції в проекті для розв'язування задачі (1.6) для вихідних даних (1.8), (1.9) та для даних (1.10). Можливий варіант тексту відповідної програми наведений далі на ліст. 3.1, де бачимо, що створена підпрограма-функція дійсно є універсальною та може використовуватися в різних проектах.

3.3.2 Розглянемо далі приклад використання окремо розробленої підпрограми-функції для визначення коренів рівняння методом половинного ділення в якості складової програмного проекту. Для цього пропонується створення проекту ROOT7.WPJ, що буде містити два джерела – файли ROOT7.FOR та ROOTS.FOR, як схематично показано на рис. 3.2. У файлі ROOT7.FOR буде міститись основна програма та функції, якими визначаються рівняння (1.6), що мають розв'язуватися (ліст. 3.2); у файлі ROOTS.FOR буде міститись окремо розроблена підпрограма-функція визначення кореня методом половинного ділення (ліст. 3.3) та, можливо, інші підпрограми, що пов'язані із визначенням коренів рівнянь.

3.3.3 В якості прикладу створення та використання статичної бібліотеки підпрограм можна розглянути створення та використання статичної бібліотеки ROOTS.LIB, яка міститиме розроблену раніше підпрограму-функцію ROOT12 (ліст. 3.3), що визначатиме корінь рівняння методом половинного ділення в заданому інтервалі із заданою похибкою. Зрозуміло, що розробка такої бібліотеки дійсно є корисною, оскільки розв'язування задачі (1.6) може знадобитися в різних проектах.

Лістинг 3.1 – Програма із функціями аргументами іншої функції

```
PROGRAM ROOT6
REAL X1,X2,ROOT
EXTERNAL F1,F2
X1=ROOT(4.0,5.0,1.0E-6,F1)
X2=ROOT(1.0,3.5,1.0E-6,F2)
PRINT*,'X1=',X1,';   F1(X1)=',F1(X1)
PRINT*,'X2=',X2,';   F2(X2)=',F2(X2)
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
REAL FUNCTION F1(X)
REAL X
F1=TAN(X)-X
END
REAL FUNCTION F2(X)
```

```
      REAL X
      F2=X**3-6*X**2+12*X-8
      END
      REAL FUNCTION ROOT(A,B,E,F)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT=(AA+BB)/2
      END
```

Лістинг 3.2 – Зміст файлу ROOT7.FOR

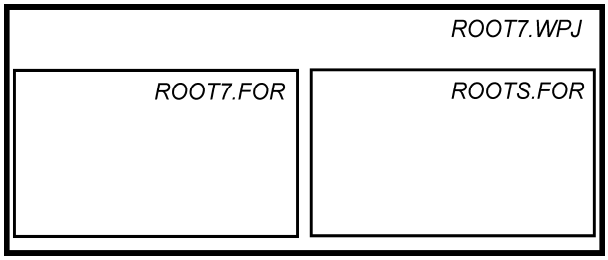


Рисунок 3.2 – Проект та його джерела

```
      PROGRAM ROOT7
      REAL X1,X2,ROOT12
      EXTERNAL F1,F2
      X1=ROOT12(4.0,5.0, 1.0E-6,F1)
      X2=ROOT12(1.0,3.5, 1.0E-6,F2)
      PRINT*,'X1=',X1,';   ','F1(X1)=',F1(X1)
      PRINT*,'X2=',X2,';   ','F2(X2)=',F2(X2)
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
      REAL FUNCTION F1(X)
      REAL X
      F1=TAN(X)-X
      END
      REAL FUNCTION F2(X)
      REAL X
      F2= X**3-6*X**2+12*X-8
      END
```

Лістинг 3.3 – Зміст файлу ROOTS.FOR

```
      REAL FUNCTION ROOT12(A,B,E,F)
      REAL A,B,E,X,FA,FX,F,AA,BB
      AA=A
```

```
      BB=B
      FA=F(AA)
100   X=(AA+BB)/2
      FX=F(X)
      IF ((FA*FX).LT.0) THEN
      BB=X
      ELSE
      AA=X
      FA=FX
      END IF
      IF ((BB-AA).GT.E) GOTO 100
      ROOT12=(AA+BB)/2
      END
```

Слід розуміти, що статична бібліотека насправді є незалежним програмним проектом. Отже, для створення цієї бібліотеки необхідно створити проект ROOTS.WPJ і в якості мети (target) проекту обрати Library[.lib], як це показано на рис. 3.3. Далі в цей проект слід додати в якості джерела файл ROOTS.FOR, наведений на ліст. 3.2, та здійснити компіляцію цього проекту. В результаті компіляції одержимо файл ROOTS.LIB, який можемо додавати до проектів в якості джерела. Для використання бібліотеки ROOTS.LIB створимо новий програмний проект ROOT8.WPJ, до якого в якості джерел додамо файл ROOT7.FOR, наведений на лістингу 3.2, та файл ROOTS.LIB бібліотеки. Далі можна здійснити компіляцію та запустити програму.

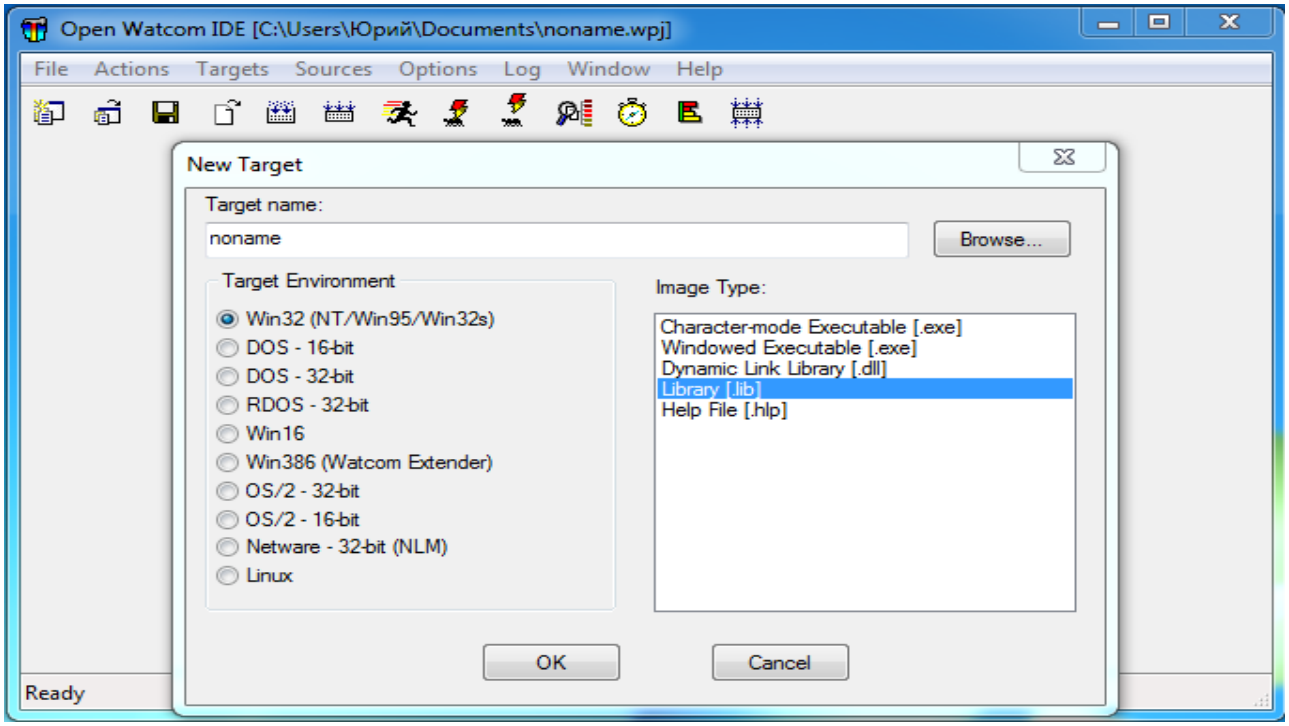


Рисунок 3.2 – Створення проекту статичної бібліотеки

3.4 Контрольні запитання та завдання

Наявність знань, що підтверджують засвоєння матеріалу заняття, можна перевірити спроможністю надати відповіді та виконати наступні питання та завдання:

1. Чи може ім’я підпрограми бути вихідним параметром для іншої підпрограми? Пояснити, яким чином це забезпечує універсальність підпрограм на прикладі визначення кореня рівняння в заданому інтервалі методом половинного ділення.

2. Яким чином в мові програмування FORTRAN здійснюється використання імені підпрограми в якості вихідного параметра для іншої підпрограми?

3. Що таке основна програма та як вона визначається в мові програмування FORTRAN? Чи можна оформляти основну програму та інші програмні одиниці в окремих файлах? Що таке проект і файл проекту, джерело і файл джерела?

4. Що таке статична бібліотека та в чому полягають її переваги перед розташуванням програмних одиниць в окремому файлі? В якому вигляді представляється статична бібліотека та як її використовувати в програмах?

5. Почитати в Інтернеті про метод хорд щодо визначення кореня рівняння у заданому інтервалі та розробити універсальну підпрограму-функцію, яка використовує цей метод. Додати цю підпрограму-функцію до бібліотеки ROOTS.LIB.

4 ВИКОНАННЯ ЗАДАНОЇ КІЛЬКОСТІ ЦИКЛІВ ПОСЛІДОВНОСТЕЙ ІНСТРУКЦІЙ

У програмах різного призначення досить часто виникає потреба виконувати задану кількість разів деяку послідовність інструкцій, отже, здійснення такого відноситься до типових задач програмування.

4.1 Теоретичні відомості про виконання заданої кількості циклів

При необхідності виконання послідовності інструкцій програми задану кількість разів принципово потрібно контролювати кількість разів, що вже були виконані. В графічних схемах алгоритмів програм передбачене спеціальне скорочене позначення циклічного виконання послідовностей інструкцій, як показано на рис. 4.1.

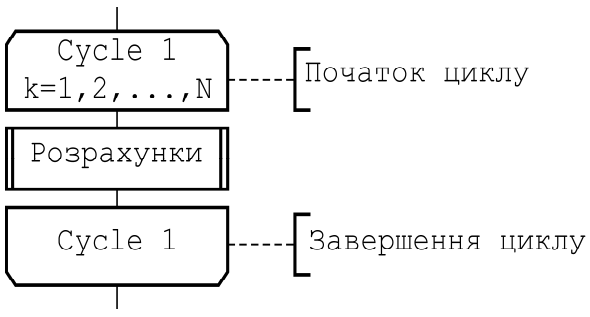


Рисунок 4.1 – Умовне позначення циклів на схемах алгоритмів

4.1.1 Виконання заданої кількості циклів повторення послідовності інструкцій взагалі можна здійснити шляхом підрахування кількості вже здійснених циклів та використання інструкцій переходу за умовою, в якій перевіряється кількість вже виконаних циклів. Такий підхід можна здійснювати шляхом використання спеціально передбаченої в програмі змінної цілочислового типу – лічильника циклу, значення якої слід збільшувати на одиницю після завершення виконання кожного циклу послідовностей інструкцій. Далі слід порівняти значення лічильника циклу із потрібною кількістю циклів і відповідно до результатів порівняння або здійснити наступний цикл, або завершити циклічне виконання відповідної послідовності інструкцій (рис. 4.2).

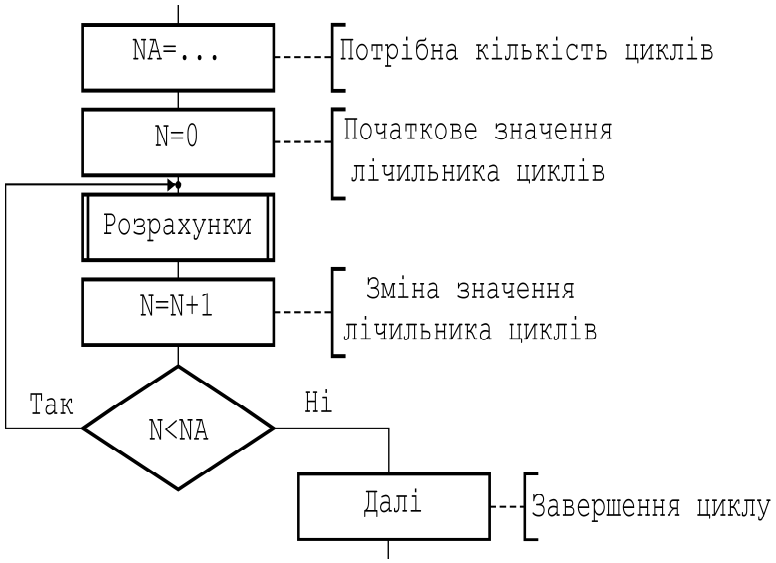


Рисунок 4.2 – Безпосереднє обчислення лічильника циклів

4.1.2 Оскільки виконання заданої кількості циклічних повторень деякої послідовності інструкцій є розповсюдженим в програмах різного призначення, то в більшості мов програмування виключно для зручності передбачають можливість організації непрямого підрахування кількості циклів, для чого передбачені відповідні спеціальні інструкції. В таких інструкціях обов'язково передбачається введення змінної – лічильника кількості циклів, та вказують необхідні початкове значення, кінцеве значення й крок зміни лічильника циклів.

В мові програмування FORTRAN циклічне виконання заданих послідовностей інструкцій записується так:

```
      ...
      DO label name = start,finish,step
      ...
      J=name
      ...
label CONTINUE
      ...
```

Послідовність інструкцій, яка має виконуватися в циклі, розпочинається одразу ж після інструкції DO та завершується інструкцією, на яку посилається мітка label інструкції DO. В інструкції DO також визначається ім'я name змінної-лічильника кількості циклів, її початкове значення start, кінцеве значення finish та крок step зміни. Для наочного розуміння тексту

програми мітка label може посилатися на спеціальну інструкцію CONTINUE, яка не виконується і введена лише, щоб визначати місця для передачі управління. Змінна циклу може бути цілою (INTEGER) або дійсною (REAL); якщо крок не вказано, то він за замовченням буде дорівнювати одиниці.

4.1.3 При створенні великих програм було підмічено, що велика кількість інструкцій переходів та міток суттєво ускладнює розуміння алгоритму програми із її тексту. Альтернативою використання міток та переходів є структурний підхід щодо розробки програм, відповідно до якого програма представляється послідовністю структур керування [2]. Стандартна мова програмування FORTRAN 77 є не дуже пристосованою до структурного програмування, тому що концепції структурного програмування були розвинені пізніше, ніж був створений стандарт мови програмування FORTRAN 77, який є результатом узагальнення досвіду багаторічного використання попередніх стандартів мови FORTRAN. В той же час фірма WATCOM внесла додаткові можливості (розширення) в компілятор FORTRAN 77, які не передбачені стандартом цієї мови та дозволяють використовувати певні концепції структурного програмування, спрямовані на додаткові можливості організації циклів тощо.

Розширенням стандартної мови програмування FORTRAN 77 є інструкція END DO, яку можна використовувати замість стандартної інструкції CONTINUE для організації циклів без використання міток (табл. 4.1). Зазначимо, що інструкція END DO передбачена в стандарті FORTRAN 90 та у сучасних стандартах мови програмування FORTRAN, тому на цю інструкцію слід звернути особливу увагу.

Таблиця 4.1 – Порівняння стандартного та структурного `DO`-циклу

Приклад стандартного `DO`-циклу	Приклад структурного `DO`-циклу

`DO 100 K=1,N`
`S=S+F(X)`
`X=X+DX`
`100 CONTINUE`
	`DO K=1,N`
	`S=S+F(X)`
	`X=X+DX`
	`END DO`

Окрім DO-циклу розширені можливості WATCOM FORTRAN 77 дозволяють реалізувати інші структури управління, серед яких доцільно згадати структури WHILE – DO – END WHILE та структури LOOP – UNTIL (табл. 4.2), аналоги яких сьогодні є обов’язково передбаченими в сучасних мовах програмування. Доведено, що використання структур керування типу IF – THEN – ELSE, DO – END DO, WHILE – DO – END WHILE та LOOP – UNTIL дозволяє написати будь-яку програму взагалі без використання міток та інструкцій переходу [2].

Таблиця 4.2 – Структурні реалізації циклу (розширені)

Приклад циклу WHILE	Приклад циклу UNTIL

WHILE (X.LT.B) DO
S=S+F(X)
X=X+DX
END WHILE
	LOOP
	S=S+F(X)
	X=X+DX
	UNTIL (X.GE.B)

Слід зазначити, що використання розширень (табл. 4.1 і табл. 4.2) мови програмування FORTRAN 77 можливе тільки за умов використання компілятора фірми WATCOM, тобто при спробі компіляції текстів програм із таким розширенням на інші компілятори, наприклад фірм IBM або Microsoft, будуть повідомлення про помилки. Відсутність структур керування в стандартному FORTRAN 77 є його суттєвим недоліком, який сприяв подальшому розвитку мови програмування FORTRAN, сучасні стандарти якої містять усі притаманні сучасним мовам програмування можливості, що забезпечують реалізацію концепції структурного програмування. В той же час, необхідність удосконалення великої кількості програм, що написані раніше на FORTRAN 77, змушує вивчати FORTRAN 77 і сьогодні. Більш того, FORTRAN 77 містить в собі узагальнення досвіду програмування з початку використання комп'ютерів і тому є зручним та корисним для початкового навчання з програмування.

4.2 Завдання щодо виконання заданої кількості циклів

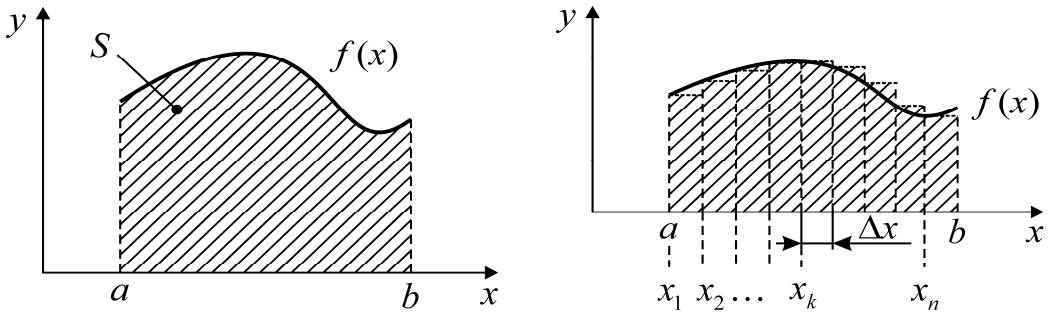
Прикладом завдання, яке вимагатиме розробки програми із заданою кількістю циклів виконання послідовностей інструкцій, є обчислення визначеного інтеграла:

$$S = \int_a^b f(x)\,dx, \qquad \ldots(4.1)$$

де a, b – задані границі інтервалу; $f(x)$ – задана функція.

Як добре відомо, геометричним образом визначеного інтеграла (4.1) є площа криволінійної трапеції, що обмежена графіком функції $f(x)$ та показана у вигляді заштрихованої області на рис. 4.3а. З урахуванням цього, розділимо інтервал інтегрування на n частин однакової довжини (рис. 4.3б):

$$\Delta x = \frac{b-a}{n}. \qquad (4.2)$$



а) *б)*

а) геометричний сенс; *б) наближене обчислення*

Рисунок 4.3 – Визначений інтеграл

Числове значення s інтеграла (2.8) наближено представимо у вигляді:

$$S \approx S_n, S_n = \sum_{k=1}^{n} f\left(x_k\right)\Delta x\,, \tag{4.3}$$

де x_k – ліва координата частини із номером k інтервалу інтегрування (див. рис. 4.3б).

Безумовно, результат розрахунку інтеграла (4.1) у вигляді (4.2), (4.3) матиме похибку, яка буде залежати від числа n; цю похибку можна геометрично уявити як суму площин незаштрихованих областей та площин областей заштрихованих прямокутників, які виходять за межі криволінійної трапеції на рис. 4.3б, що ілюструє невисоку відносну похибку наближеної формули (4.3) при обчисленні визначених інтегралів за умови, якщо число n є невеликим. Зрозуміло, що при збільшенні числа n похибка обчислення інтеграла (4.1) за формулою (4.2), (4.3) буде зменшуватися, і при достатньо великому числі n така похибка буде досить малою. Отже, програма наближеного обчислення значення визначеного інтеграла (4.1) за формулою (4.2), (4.3) зводиться до обчислення суми із великою кількістю схожих доданків, що можна реалізувати за схемою алгоритму, що представлений вище на рис. 4.2.

Розглядатимемо далі приклад програми обчислення інтеграла (4.1) за формулою (4.2), (4.3) для вихідних даних:

$$f\left(x\right) = x^3, a = 0, b = 1\,. \tag{4.4}$$

Інтеграл (4.1) для вихідних даних (4.4) можна підрахувати точно:

$$S = 0,25\,. \tag{4.5}$$

Шляхом порівняння одержаного з використанням відповідної програми за наближеною формулою (4.2), (4.3) результату обчислення інтеграла (4.1) для вихідних даних (4.4) із його точним значенням (4.5) можна оцінити похибку обчислення інтеграла та дослідити зв’язок між кількістю інтервалів n та цією похибкою.

4.3 Приклади виконання заданої кількості циклів

Покажемо приклади виконання в програмах заданої кількості циклів послідовностей інструкції для обчислення визначеного інтеграла.

4.2.1 Приклад програми, в якій передбачено виконання заданої кількості циклів послідовностей інструкції шляхом використання безпосереднього лічильника циклів, наведений на ліст. 4.1. Зауважимо, що в наведеному прикладі (ліст. 4.1) обчислення визначеного інтеграла оформлене як окрема підпрограма-функція, що цілком узгоджується із сенсом такого обчислення.

4.2.2 Приклад підпрограми-функції, в якій для обчислення інтеграла передбачено виконання заданої кількості циклів послідовностей інструкції, яке реалізоване за допомогою непрямого лічильника циклів, наведений на ліст. 4.2. Бачимо, що використання непрямого лічильника циклів дозолило скоротити текст підпрограми обчислення інтеграла на 20 % порівняно із використанням безпосереднього лічильника циклів.

Лістинг 4.1 – Безпосередній лічильник циклів та його використання для наближеного обчислення визначеного інтеграла

```
      PROGRAM INTEGRATION1
      REAL S,F,INTEGRAL1
      EXTERNAL F
      S=INTERGRAL1(0.0,0.1,150,F)
      PRINT*,'S=',S
      PAUSE 'PRESS ENTER TO EXIT'
      END
      REAL FUNCTION F(X)
      REAL X
      F=X**3
      END
      REAL FUNCTION INTEGRAL1(A,B,N,F)
      REAL S,DX,X,A,B,F
      INTEGER N,K
      DX=(B-A)/N
      K=0
      S=0
      X=A
100   S=S+F(X)
      K=K+1
      IF (K.LT.N) THEN
      X=X+DX
      GOTO 100
```

```
      END IF
      INTEGRAL1=S*DX
      END
```

Лістинг 4.2 – Непрямий лічильник циклів та його використання для наближеного обчислення визначеного інтеграла

```
      REAL FUNCTION INTEGRAL1(A,B,N,F)
      REAL S,DX,X,A,B,F
      INTEGER N,K
      DX=(B-A)/N
      S=0
      X=A
      DO 100 K=1,N
      S=S+F(X)
      X=X+DX
100   CONTINUE
      INTEGRAL1=S*DX
      END
```

4.4 Контрольні запитання та завдання

Наявність знань, що підтверджують засвоєння матеріалу заняття, можна перевірити спроможністю надати відповіді та виконати наступні питання та завдання:

1. Як позначається на графічних схемах алгоритмів програм циклічне виконання послідовності інструкцій задану кількість разів?
2. Навести схему алгоритму фрагменту програми, в якому здійснюється виконання заданої кількості циклів послідовності інструкцій з використанням безпосереднього лічильника кількості циклів.
3. Пояснити, як в мові програмування FORTRAN реалізується виконання заданої кількості циклів за допомогою непрямого лічильника кількості циклів.
4. Навести структурну форму непрямого лічильника кількості циклів, що передбачені в розширеннях мови програмування FORTRAN.
5. Модифікувати текст підпрограми, наведений вище на ліст. 4.2, таким чином, щоб необхідний цикл реалізовувався із використанням структурної інструкції, яка наведена вище в табл. 4.1.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ursula Gertrud von der Leyen. A Union that strives for more. My agenda for Europe: Political Guidelines for the Next European Commission 2019-2024. Available at:
https://commission.europa.eu/document/download/063d44e9-04ed-4033-acf9-639ecb187e87_en?filename=political-guidelines-next-commission_en.pdf [Accessed: 15.04.2023].
2. Dahl O.-J. Structured programming / O.-J. Dahl, E.W. Dijkstra, C.A.R. Hoare. – London-New York: Academic Press, 1973. – 228 p.
3. Chapman S.J. Fortran for Scientists and Engineers / S.J. Chapman. – New York: McGrawHill Education, 2018. – 1024 p.
4. Markus A. Modern Fortran in Practice / A. Markus. – New York: Cambridge University Pres, 2012. – 253 p.
5. http://openwatcom.org
6. https://github.com/open-watcom/open-watcom-1.9

Для нотаток

Електронне навчальне видання комбінованого використання
Можна використовувати в локальному та мережному режимі

Коробов Валерій Іванович
Ромашов Юрій Володимирович
Стєпанова Катерина Вадимівна

ЕЛЕМЕНТИ СТРУКТУРНОГО ПРОГРАМУВАННЯ

Методичні рекомендації
до практичних занять з курсу «Програмування»
для студентів спеціальностей 113 «Прикладна математика»,
142 «Енергетичне машинобудування», 143 «Атомна енергетика»,
174 «Автоматизація, комп'ютерно-інтегровані технології
та робототехніка»

Коректор *О. В. Анцибора*
Комп'ютерне верстання *В. В. Савінкова*

Підписано до розміщення 17.06.2023. Гарнітура Times New Roman.
Ум. друк. арк. 2,17. Обсяг 0,98 Мб. Зам. 117/23

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна