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

ПРОГРАМУВАННЯ

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

Електронне видання

Харків – 2023

УДК 519.6: 519.7: 621.039: 681.3
К 68

Рецензенти:

С. Ю. Ігнатович – доктор фізико-математичних наук, професор кафедри прикладної 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.)

Коробов В. І.

К 68 Програмування : методичні рекомендації до практичних занять для студентів спеціальностей 113 «Прикладна математика», 142 «Енергетичне машинобудування», 143 «Атомна енергетика», 151 «Автоматизація та комп'ютерно-інтегровані технології» [Електронне видання] / В. І. Коробов, Ю. В. Ромашов, К. В. Стєпанова, М. О. Бебія. – Харків : ХНУ імені В. Н. Каразіна, 2023. – (PDF 28 с.)

Методичні вказівки підготовлені щодо вивчення теми «Загальні поняття, зв'язані із програмуванням» відповідно до основних положень програм з навчальної дисципліни «Програмування». Обговорюються створення простішої програми, виведення та введення даних системними пристроями, а також можливості управління порядком виконання інструкцій.

Методичні вказівки призначені для студентів різних курсів та рівнів вищої освіти спеціальностей 113 «Прикладна математика», 142 «Енергетичне машинобудування», 143 «Атомна енергетика», 151 «Автоматизація та комп'ютерно-інтегровані технології».

УДК 519.6: 519.7: 621.039: 681.3

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

Коробов Валерій Іванович
Ромашов Юрій Володимирович
Стєпанова Катерина Вадимівна
Бебія Максим Отарійович

ПРОГРАМУВАННЯ

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

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

Підписано до розміщення 17.06.2023. Гарнітура Times New Roman.
Ум. друк. арк. 1,67. Обсяг 0,388 Мб. Зам. 115/23.

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

ЗМІСТ

ВСТУП

Програмування, як і кожна галузь знань, спирається на сукупність деяких базових загальних понять та концепцій, саме з використанням яких формуються усі основні результати. На відміну від багатьох інших галузей знань програмування цифрових електронно-обчислювальних машин, які сьогодні прийнято називати комп'ютерами, є відносно новою галуззю, що має порівняно коротку історію, яка сьогодні не перевищує ста років. В той же час, навіть за таку малу історію, програмування досить сильно розвинулось, але надсучасні технології та стилі програмування досить суттєво ґрунтуються на базових загальних поняттях та концепціях. Отже, подальше формування знань щодо сучасних технологій та мов програмування вимагає досить стійкого володіння базовими загальними поняттями та концепціями щодо програм, мов програмування та їх інструкцій. На таких базових загальних поняттях та концепціях в програмуванні, як програма, мова програмування та її інструкції, істотно ґрунтуються основні прийоми програмування для розв'язування типових завдань щодо розробки програм.

Метою цих методичних вказівок до практичних занять з курсу програмування є засвоєння теоретичного матеріалу щодо таких базових загальних понять та концепцій програмування, як програма, мова програмування та її інструкції, а також формування практичних навичок використання основних прийомів програмування для розв'язування типових завдань щодо розробки програм.

Реалізація зазначеної мети буде здійснюватися шляхом поступового створення від простіших до більш складних програм для виконання інженерних розрахунків щодо визначення товщини стінки корпусу ядерного реактора із умови міцності. В якості мови 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 удосконалених версій FORTRAN щодо розробки програм для наукових та інженерних розрахунків [1, 2].

1 СТВОРЕННЯ ПРОСТІШОЇ ПРОГРАМИ ДЛЯ РОЗРАХУНКУ

Комп'ютери та наявні в них програми є дуже розповсюдженими сьогодні і використовуються навіть у побуті, але історично первинним є використання комп'ютерів та програмування наукових та інженерних розрахунків, які вимагають великих обсягів обчислень. Слід розуміти, що програмування наукових та інженерних розрахунків і сьогодні є важливою галуззю впровадження прикладної математики, тому саме такі розрахунки використовуватимемо далі в основі прикладів завдань щодо створення відповідних програм для засвоєння базових прийомів програмування.

1.1 Теоретичні відомості щодо створення простіших програм

Можливості комп'ютерів обумовлені не тільки високою швидкістю виконання дій та великими обсягами пам'яті для зберігання проміжних результатів, а насамперед автоматизацією обчислювань на основі програмного управління, яке здійснюється відповідно програмі, яка зберігається у пам'яті комп'ютера разом із даними.

1.1.1 Програму можна уявляти як послідовність команд, які можуть виконуватися комп'ютером для одержання певних результатів. За рахунок відповідного вибору команд та їхньої послідовності у програмі можна розв'язувати широке коло задач. Зрозуміло, що команди, які містяться у програмах, природно мають бути виключно із множини команд, які відомі процесору комп'ютера. Сучасні комп'ютери містять сотні команд, які є примітивними, що суттєво ускладнює написання програм безпосередньо на машинному коді. Для збільшення швидкості написання програм було запропоновано використовувати мови програмування, які представляють сукупність правил символьного запису програм, що дозволяють представляти програми у зрозумілому наочному текстовому вигляді. За допомогою спеціальних програм – компіляторів (трансляторів) програми, написані на мовах програмування, перетворюються у відповідні їм послідовності команд процесора, тобто у програми, що безпосередньо виконуються на комп'ютері. Якщо послідовність дій, що реалізується в програмі, уявляти алгоритмом, то програма буде записом цього алгоритму.

Різноманіття завдань для комп'ютерних програм природно породжує різноманіття мов програмування, кожна з яких є більш зручною та пристосованою до створення програм певного класу. Історично першою задачею, для розв'язування якої створювалися програми, було виконання математичних обчислень за заданими алгоритмами, для чого було створено мову програмування FORTRAN, назва якої відповідна словосполученню FORmula TRANslation [3]; мова програмування FORTRAN є найстарішою

із розповсюджених сьогодні мов програмування [1, 2]. Більш сучасні мови програмування, такі як C, Pascal, Basic, PROLOG, LISP, Python, створювалися з урахуванням досвіду використання FORTRAN як більш універсальні або більш спеціалізовані, тому такі мови недоцільно використовувати при первинному знайомстві із програмуванням взагалі, для чого найбільш зручним є саме FORTRAN 77.

1.1.2 Для запису програм на мові програмування FORTRAN передбачено використання виключно доступних на стандартній комп'ютерній клавіатурі наступних символів [3–5]: 26 великих літер латиниці (A B C D … X Y Z), 10 цифр (0 1 2 3 4 5 6 7 8 9) та 13 спеціальних символів (␣ . , = + - * / ( ) : ‘ $), причому перший із 13 спеціальних символів означає пробіл, а більшість інших – зрозуміло із математики. Текст програми розміщується у рядках таким чином, що кожен рядок містить одну інструкцію, а послідовність виконання інструкцій визначається їхнім розташуванням у рядках, таким чином, що дії виконуються послідовно від верхніх рядків до нижніх до закінчення програми [3–5]. Текст програми у кожному рядку слід починати після 6 пробілів, які зарезервовано для розміщення службової інформації програми. Текст програми можна розташовувати в рядку впритул до 74 символу, місце після якого зарезервоване для розміщення необов'язкових коментарів вільної форми щодо тексту програми для зручного його використання.

При виконанні навчальних завдань будемо використовувати вільне для користування програмне забезпечення на основі компілятора фірми WATCOM, яке є загальнодоступним на сайті [6]. Кожна із програм буде являти собою окремий проект, що має містити як мінімум два файли, які доцільно зберігати на диску в окремій папці таким чином, щоб ім'я папки для зберігання проекту збігалося з ім'ям проекту. Перший із означених файлів – це файл проекту із розширенням *.WPJ (тобто WatcomProJect), що містить інформацію про інші файли, з яких складається проект. Далі будемо використовувати проекти, що складаються із файлів-джерел (sources), що містять програми, написані на мові програмування FORTRAN та відповідно до цього мають зрозуміле розширення *.FOR.

Кожна програма у FORTRAN має розпочинатися з інструкції PROGRAM, після якої слід вказати ім'я програми, і завершуватися інструкцією END в останньому рядку тексту програми. Дані, які використовуються у програмі можна зберігати у вигляді змінних. Змінна являє собою об'єкт програми, який характеризується типом, ім'ям та значенням. При виконанні математичних розрахунків можна використовувати тип змінної REAL, який представляє звичайні дійсні числа. Імена змінних мають починатися із літери та можуть містити досить велику кількість літер та цифр, але не можуть містити спеціальні символи. Типи усіх змінних програми визначають на початку програми наступним чином:

type *list*, (1.1)

де type – ідентифікатор типу даних, наприклад REAL; *list* – список розділених комами імен змінних програми, що мають тип type .

Не обов'язково визначати усі змінні одного типу в одному рядку, тобто можна передбачити декілька рядків вигляду (1.1) для визначення змінних одного типу.

Значення змінної визначають в тексті програми в окремому рядку наступним чином:

NAME = *value*, (1.2)

де NAME – ім'я та *value* – значення змінної, записане звичайними цифрами з використанням точки для відділення дрібної частини, або вираз, записаний з використанням математичних операцій.

За допомогою змінних та числових констант можна визначати різноманітні змінні, а за допомогою виразів, які містять математичні операції + - * / та дужки (), виконувати також математичні обчислення.

В мовах програмування окрім числових даних передбачається також використання текстових даних, які, зазвичай, визначають у вигляді набору символів, що обмежені прямими одинарними лапками ‘.

Для виведення на екран монітора значень змінних у FORTRAN передбачена спеціальна інструкція:

PRINT*, *list*, (1.3)

де * – позначка автоматичного форматування; *list* – список розділених комами імен змінних, констант або виразів, значення яких слід відобразити на екрані монітора.

Для забезпечення можливості введення значень з клавіатури в FORTRAN передбачена спеціальна інструкція:

READ*, *list*, , (1.4)

де * – позначка автоматичного форматування; *list* – список розділених комами імен змінних, яким мають бути присвоєні введені з клавіатури значення.

Для призупинення роботи програми із можливістю подальшого її продовження в FORTRAN передбачена спеціальна інструкція:

PAUSE *textdata*, (1.5)

де *textdata* – необов'язковий параметр, який представляє текстові дані, що будуть виведені на монітор при виконанні інструкції.

При виконанні інструкцій (1.4) та (1.5) робота програми тимчасово призупиняється та продовжується лише після натискання клавіші Enter на клавіатурі.

1.2 Завдання щодо створення простішої програми

Простіша програма виконує розрахунки за заданими формулами. В якості прикладу розглядатимемо визначення мінімально допустимої умовами міцності товщини стінки циліндричного корпусу ядерного реактора, яка відповідно нормам розрахунку [7] визначається так:

$$S_R = \frac{pD}{2[\sigma] - p}, \tag{1.6}$$

де S_R (мм) – мінімальна товщина стінки циліндричного корпусу ядерного реактора; p (МПа) – внутрішній тиск; D (мм) – внутрішній діаметр та $[\sigma]$ (МПа) – допустиме напруження матеріалу корпусу.

Слід зазначити, що створення навіть простих програм, які виконують нескладні розрахунки за заданою формулою, наприклад (1.6), є досить корисним, оскільки використання таких програм значно прискорює багаторазове виконання відповідних розрахунків.

1.3 Приклади простіших програм

Розглянемо далі приклади простіших програм, які ілюструють базові прийоми програмування розрахунків за заданою формулою (1.6). Для цього створимо проект EP1.WPJ, який буде містити створений на мові FORTRAN 77 файл-джерело EP1.FOR.

1.3.1 Текст найпростішої програми, яка виконує розрахунок за заданою формулою (1.6), має наступний вигляд:

Лістинг 1.1 – Зміст файлу EP1.FOR програми, в якій здійснюється розрахунок за заданою формулою

```
PROGRAM EP1
REAL SR,P,D,SIGMA
P=10.0
D=500.0
SIGMA=250.0
SR=P*D/(2*SIGMA-P)
END
```

Текст програми (ліст. 1.1) оформлений відповідно до вимог мови програмування FORTRAN та розпочинається з інструкції PROGRAM, після якої вказано ім’я програми, і завершується інструкцією END. На початку кожного рядка передбачено шість пробілів, які зарезервовані для службової інформації програми, але у наведеному прикладі програма не

містить службової інформації, отже, маємо просто шість пробілів на початку кожного рядка.

В наведеній програмі (ліст. 1.1) за допомогою інструкції вигляду (1.1) передбачено використання змінних SR,P,D,SIGMA, які мають тип REAL, відповідний дійсним числам. Далі змінним за допомогою інструкції вигляду (1.2) присвоюються необхідні значення і здійснюється обчислення за формулою (1.5), що представлена в програмі у відповідному вигляді.

1.3.2 Недоліком програми, що наведена вище на ліст. 1.1, є те, що користувач не побачить на моніторі жодних результатів розрахунків, тобто насправді взагалі не є корисною. Позбавлена цього недоліку програма може мати наступний вигляд:

Лістинг 1.2 – Зміст файлу EP1.FOR програми, в якій здійснюється розрахунок за формулою, друк результату та призупинення

```
PROGRAM EP1
REAL SR,P,D,SIGMA
P=10.0
D=500.0
SIGMA=250.0
SR=P*D/(2*SIGMA-P)
PRINT *,SR
PAUSE
END
```

В програмі (ліст. 1.2) за дорогою інструкції вигляду (1.3) передбачено відображення на екрані монітора значення змінної SR, яка представляє результат розрахунку, в форматі, що прийнятий за замовчуванням завдяки наявності символу "*". Також за допомогою інструкції вигляду (1.5) передбачено (ліст. 1.2) призупинення роботи програми, доки не буде натиснута клавіша Enter на клавіатурі комп'ютера, що дозволяє користувачу побачити результати розрахунків та продовжити роботу.

1.3.3 Недоліком програми, що наведена на ліст. 1.2, є те, що користувач не може змінювати значення вихідних даних при виконанні програми, тобто для виконання розрахунку з іншими вихідними даними слід кожного разу змінювати відповідним чином текст програми та здійснювати її компіляцію. Хоча в розглянутому прикладі (ліст. 1.2) необхідні модифікації програми є нескладними та зводяться до визначення нових значень змінних, але у випадку більш складних програм відповідні модифікації для зміни вихідних даних розрахунків можуть вимагати досить високої кваліфікації користувача в галузі програмування.

Отже, що необхідність зміни вихідних даних без модифікації тексту програми та її повторної компіляції значно підвищить ефективність використання програм та дозволить також значно розширити кількість корис-

тувачів програми, оскільки позбавить користувачів програм необхідності мати компетенції в галузі програмування для внесення необхідних змін у програми. Остання обставина є дуже важливою, оскільки програми створюються саме для підвищення ефективності виконання розрахунків користувачами, а модифікація програми може займати певний час та містить при цьому ризики ненавмисного пошкодження тексту програми недосвідченими користувачами. Програма для виконання розрахунків за формулою (1.6), в якій передбачається можливість зміни вихідних даних без модифікації тексту програми, має наступний вигляд:

Лістинг 1.3 – Зміст файлу EP1.FOR програми, в якій здійснюється можливість введення даних розрахунку

```
PROGRAM EP1
REAL SR,P,D,SIGMA
READ *,P
READ *,D
READ *,SIGMA
SR=P*D/(2*SIGMA-P)
PRINT *,SR
PAUSE
END
```

В програмі (ліст. 1.3) замість присвоювання значень відповідних змінних використано інструкцію вигляду (1.4), яка призупиняє роботу програми для введення з клавіатури значення вказаної в ній змінної, а після завершення введення та натискання клавіші Enter продовжує виконання програми. Завдяки цьому користувач має можливість вводити необхідні йому дані для виконання розрахунку, і програма набуває властивості універсальності.

1.3.4 Робота із показаною на ліст. 1.3 програмою передбачає, що користувач знає, в якому порядку слід вводити вихідні дані, та розуміє, що останнє число, яке виведе комп’ютер на екрані монітора, буде результатом розрахунків; користувач також має знати, що після ознайомлення із результатом розрахунку для завершення роботи програми слід натиснути клавішу Enter на клавіатурі. Користування такою програмою потребує від користувача ретельного вивчення її алгоритму. Зрозуміло, що роботу із цією програмою можна суттєво полегшити шляхом виведення на екран монітора відповідних коментарів, що підказуватимуть користувачу, що саме він має роботи. Коментарі до програми являють собою текст, а точніше набір символів. Отже, маємо можливість суттєво удосконалити програму (ліст. 1.3) наступним чином:

Лістинг 1.4 – Зміст файлу EP1.FOR удосконаленої універсальної програми для виконання розрахунку

```
PROGRAM EP1
REAL SR,P,D,SIGMA
PRINT *,'INPUT P(MPA):'
READ *,P
PRINT *,'INPUT D(MM):'
READ *,D
PRINT *,'INPUT SIGMA(MPA):'
READ *,SIGMA
SR=P*D/(2*SIGMA-P)
PRINT *,'SR=',SR
PAUSE 'PRESS ENTER TO EXIT PROGRAM'
END
```

Завдяки текстовим константам ‘INPUT P(MPA):’, ‘INPUT D(MM):’, ‘INPUT SIGMA(MPA):’, а також ‘PRESS ENTER TO EXIT PROGRAM’, які виводяться на екран монітора протягом роботи програми (ліст. 1.4), значно спрощується робота користувача із програмою – користувач лише відповідає на питання комп’ютера і не повинен ретельно вивчати алгоритм, який реалізує програма.

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. Що таке програма?
2. Що таке мова програмування?
3. Що таке компілятор?
4. Які інструкції мають бути у першому та останньому рядках програми на мові FORTRAN?
5. Що таке змінна та як визначається її тип та значення? Які бувають типи змінних?
6. Які інструкції використовуються для виведення та введення даних?
7. Яка інструкція використовується для тимчасового призупинення виконання програми?
8. Напишіть програму обчислення об’єму циліндра, який визначається за формулою:

$$V = \pi R^2 h, \tag{1.7}$$

де V – об’єм циліндра; R – радіус циліндра; h – висота циліндра.

2 ВИВЕДЕННЯ ТА ВВЕДЕННЯ ДАНИХ СИСТЕМНИМИ ПРИСТРОЯМИ

В операційній системі комп'ютерів обов'язково передбачаються різноманітні пристрої, призначені для виведення та введення даних. Наприклад, до сучасних персональних комп'ютерів обов'язково підключають монітор, клавіатуру та маніпулятор (мишу), які взагалі є універсальними системними пристроями для виведення та введення даних. В кожній мові програмування для виведення та введення даних з використанням системних пристроїв передбачені відповідні можливості. Деякі особливості передбачених в мові програмування FORTRAN можливостей щодо виведення та введення даних за допомогою системних пристроїв є типовими для багатьох інших мов програмування, у тому числі C та Pyuthon, тому представляють значний інтерес.

2.1 Теоретичні відомості щодо виведення та введення даних

В мові програмування FORTRAN передбачені інструкція PRINT та READ для виведення та введення даних за допомогою системних пристроїв – монітора та клавіатури відповідно. Виведення та введення даних здійснюється у символьному вигляді. Розглянутий вище вигляд (1.3) та (1.4) інструкції виведення та введення відповідає їхнім окремим найпростішим випадкам, а далі розглянемо загальну форму інструкцій READ та PRINT.

2.1.1 Інструкція виведення даних з клавіатури у загальному випадку має вигляд:

$$\text{PRINT } fmt, outlist, \tag{2.1}$$

де *fmt* – опис формату, який визначає правило перетворення значень із списку виведення у набір символів для представлення на екрані монітора, а *outlist* – список виведення у вигляді розділених комами імен змінних, констант або виразів, значення яких слід відобразити на екрані монітора.

Інструкція введення даних з клавіатури у загальному випадку має вигляд:

$$\text{READ } fmt, inplist, \tag{2.2}$$

де *fmt* – опис формату, який визначає правило перетворення введених з клавіатури символів у значення змінних, що містяться у списку введення, а *inplist* – список введення у вигляді розділених комами імен змінних, яким мають бути присвоєні введені з клавіатури значення.

Розглянуті раніше інструкції вигляду (1.3) та (1.4) насправді є окремими випадками інструкцій (2.1) та (2.2) загального вигляду, в яких передбачено вільний формат виведення та введення даних, відповідний наступному визначенню формату:

$$fmt = *. \tag{2.3}$$

Використання вільного формату виведення та введення є найпростішим, але слід пам'ятати, що форма виведення та правила перетворення даних можуть відрізнятися на різних комп'ютерах у випадку використання вільного формату.

2.1.2 Для забезпечення однакового виведення та введення даних програмою при її роботі на різних комп'ютерах передбачене точне визначення форматів виведення та ведення; правила визначення форматів виведення та введення є однаковими.

Формат виведення/введення являє собою текстову константу (або змінну текстового типу, що буде розглянута пізніше), в якій спеціальним чином закодовані правила перетворення значень змінних у набір символів та навпаки. Кількість та послідовність визначення таких правил має відповідати кількості змінних та їхній послідовності у списку виведення/ введення. Кожне із таких окремих правил перетворення записується спеціальним чином у вигляді дескриптора – число-літерного коду, який визначає відповідне правило перетворення, а дескриптори, що відповідають різним змінним із спуску введення, розділяються комами. Кожен із дескрипторів має бути узгодженим із типом даних, який він перетворює, та із формою представлення таких даних у символьному вигляді. Текстова константа, що визначає формат введення або (та) виведення, обов'язково має починатися з символу "(" та завершуватися символом ")".

При введені та виведенні дійсних чисел, коли відомий порядок їхніх величин, зручно використовувати наступний загальноприйнятий формат чисел:

$$\underbrace{\pm NN\ldots N.\underbrace{NN\ldots N}_{d}}_{w}, \tag{2.4}$$

де $\pm$ – знак числа; $NN\ldots N$ – деяка послідовність цифр; "." – десятинна крапка; d – кількість значущих цифр після десятинної крапки; w – загальна кількість символів для представлення числа, включаючи знак, цифри та десятинну крапку.

Для введення та виведення дійсних чисел у вигляді (2.4) у мові програмування FORTRAN передбачені дескриптори F [3–5], які мають наступний вигляд:

$$Fw.d, \tag{2.5}$$

де w – ціле число, яке вказує загальну кількість символів, що мають використовуватися для представлення числа, включаючи знак, цифри та десятинну крапку; d – ціле число, що вказує кількість цифр після десятинної крапки.

Більш універсальним є представлення дійсних чисел у наступному форматі:

$$\pm 0.\underbrace{NN\ldots N}_{d}\cdot 10^{\pm NN} \equiv \underbrace{\pm 0.\underbrace{NN\ldots N}_{d} E \pm NN}_{w}, \tag{2.6}$$

де $\pm$ – знак числа та десятинного ступеня; *NN....N* – деяка послідовність цифр; *NN* – послідовність двох цифр; "." – десятинна крапка; *d* – кількість цифр після десятинної крапки; *w* – загальна кількість символів для представлення числа, включаючи знак, цифри, десятинну крапку, символ *E* – ознаку десятинного ступеня та два числа, що визначають десятинний ступінь.

Для введення та виведення дійсних чисел у вигляді (2.6) у мові програмування FORTRAN передбачені дескриптори E [3–5], які мають наступний вигляд:

$$Ew.d, \tag{2.7}$$

де w – ціле число, що вказує загальну кількість символів для представлення числа, включаючи знаки числа та десятинного ступеня, десятинну крапку, а також символ-ознаку десятинного ступеня та три числа для визначення десятинного ступеня; d – ціле число, що вказує кількість цифр після десятинної крапки.

Для форматного введення та виведення також корисним є дескриптор X, який дозволяє пропускати задану кількість символів при введенні або задавати задану кількість пробілів при виведенні даних та має вигляд [3–5]:

$$wX, \tag{2.8}$$

де w – ціле число, що вказує, яку кількість символів слід пропустити при введенні та скільки пробілів слід надрукувати у рядку при виведенні.

Введення та виведення даних доцільно здійснювати після розміщення на екрані монітора необхідних коментарів, для чого може бути корисним дескриптор A [3–5], який має наступний вигляд:

$$Aw, \tag{2.9}$$

де w – розмір поля текстових даних.

Допускається не вказувати розмір поля в (2.9), тоді розмір поля визначається автоматично відповідно до розміру поля введеного об’єкта

або даних для виводу. Корисним також є недрукований дескриптор \, який має відокремлюватися від інших дескрипторів за допомогою коми та ставитися наприкінці опису формату перед символом ")", якщо немає потреби до переходу на новий рядок. Така потреба буває необхідна для забезпечення зручності при введені вихідних даних, бо дозволяє розташовувати коментар до даних, що необхідно ввести, та введення цих даних в одному рядку. Слід підкреслити, що дескриптор \ є нестандартним та притаманним насамперед діалекту WATCOM мови програмування FORTRAN; в інших діалектах FORTRAN замість дескриптора \ використовують еквівалентний йому дескриптор $.

2.1.3 В програмах іноді приходиться використовувати велику кількість однакових описів форматів для введення та виведення, тому іноді буває зручним окремо описати формат введення або виведення та посилатися на це описання у різних місцях програми. Для цього передбачена не виконувана інструкція опису формату, яка має вигляд:

label FORMAT FMTC, , (2.10)

де label – унікальна мітка, яка дозволяє послатися на формат та являє собою цифри, які мають бути розташованими у рядку відповідної інструкції з 1-го по п’ятий символи (старші нулі та пробіли ігноруються); FMTC – опис формату, представлений змістом символьного опису відповідного формату, тобто без лапок.

Зазначимо, що опис формату у вигляді (2.10) може бути розташованим у будь-якому місці програми, але такі описи зазвичай розташовують наприкінці програми перед інструкцією END.

2.2 Приклади виведення/введення в програмах

Розглянемо низку прикладів програм, які використовують виведення та введення даних та коментарів.

2.2.1 Приклад використання вільного формату введення та виведення в програмі розглянемо у наступному вигляді:

Лістинг 2.1 – Приклад введення та виведення у вільному форматі

```
PROGRAM TESTREADPRINT
REAL A,B,C
READ *,A,B,C
PRINT *,'A=',A, 'B=',B,'C=',C
PAUSE 'PRESS ENTER TO EXIT THE PROGRAM'
END
```

Вільна форма введення та виведення визначена в ліст. 2.1 шляхом визначення формату у вигляді (2.3). Список введення містить імена змінних A, B, C, значення яких мають бути введені з клавіатури через коми або(та) пробіли. Список для виведення містить символьні константи ‘A=‘, ‘B=‘, ‘C=‘ та змінні A, B, C, які будуть виведені на екран монітора в окремому рядку. Рекомендується декілька разів запустити програму (ліст. 2.1) та здійснити різноманітні введення і подивитись відповідні їм одержувані результати; спробуйте вводити одні й ті ж числові дані різними способами, насамперед у звичайному вигляді (2.4) та в універсальному вигляді (2.6).

2.2.2 Приклади використання дескрипторів (2.5) та (2.7), (2.8) в інструкціях (2.1) та (2.2) для форматного введення та виведення у програмі наведені у простішій програмі, в якій здійснюється введення даних та виведення введених даних на екран монітора:

Лістинг 2.2 – Приклад форматного введення та виведення

```
PROGRAM TESTFORMATREADPRINT
REAL P1,P2
READ ‘(F6.2,1X,E15.8)’,P1,P2
PRINT ‘(F6.2,1X,E15.8)’,P1,P2
PAUSE ‘PRESS ENTER TO EXIT THE PROGRAM’
END
```

Слід зазначити, що користуватися програмами із форматним вводом при використанні дескриптора E слід дуже обережно, оскільки помилки при введенні можуть привести до некоректної роботи програми. Слід рекомендувати для введення даних використовувати дескриптори F, а дескриптори E слід використовувати переважно для виведення даних.

2.2.3 Розглянемо використання дескриптора (2.9) та нестандартного дескриптора \, що допомагають представляти інформацію на екрані монітора у зручному для користувача вигляді. При роботі програми, текст якої представлено в ліст. 2.3, введення даних та виведення результатів є дуже зручним, оскільки користувач бачить, яку саме величину він має вводити та в якій розмірності, а при виведенні – має можливість проконтролювати введені дані та результат розрахунку із зазначенням розмірності.

Лістинг 2.3 – Приклад побудови зручного введення та виведення

```
PROGRAM SR
REAL SR,P,D,SIGMA
PRINT’(A,\)’,’INPUT PRESSURE, P(MPA):’
READ’(F6.2)’,P
PRINT’(A,\)’,’INPUT DIAMETER, D(MM):’
READ’(F6.2)’,D
PRINT’(A,\)’,’INPUT PERM STRESS, SIGMA(MPA):’
```

```
      READ ‘(F6.2)’,SIGMA
      SR=P*D/(2*SIGMA-P)
      PRINT’(A,F6.2,1X,A)’,’P=‘,P,’(MPA)’
      PRINT’(A,F6.2,1X,A)’,’D=‘,D,’(MM)’
      PRINT’(A,F6.2,1X,A)’,’SIGMA=‘,SIGMA,’(MPA)’
      PRINT’(A,E15.8,1X,A)’,’SR=‘,SR,’(MM)’
      PAUSE ‘PRESS ENTER TO EXIT PROGRAM’
      END
```

2.2.4 Приклад використання окремого опису формату у вигляді (2.10) в програмі може мати наступний вигляд:

Лістинг 2.4 – Приклад використання окремого опису формату

```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT 100,’INPUT PRESSURE, P(MPA):’
      READ’(F6.2)’,P
      PRINT 100,’INPUT DIAMETER, D(MM):’
      READ’(F6.2)’,D
      PRINT 100,’INPUT PERM STRESS, SIGMA(MPA):’
      READ ‘(F6.2)’,SIGMA
      SR=P*D/(2*SIGMA-P)
      PRINT 110,’P=‘,P,’(MPA)’
      PRINT 110,’D=‘,D,’(MM)’
      PRINT 110,’SIGMA=‘,SIGMA,’(MPA)’
      PRINT’(A,E15.8,1X,A)’,’SR=‘,SR,’(MM)’
100   FORMAT (A,\)
110   FORMAT (A,F6.2,1X,A)
      PAUSE ‘PRESS ENTER TO EXIT PROGRAM’
      END
```

Бачимо (ліст. 2.4), що окремо визначені формати використовують тричі у тексті програми.

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 системні пристрої введення та виведення?
2. Який загальний вигляд мають інструкції виведення та введення даних?
3. Як визначити вільний формат виведення/введення?
4. Як здійснюється опис формату виведення/введення?

5. Які дескриптори використовують для виведення/введення чисел?
6. Які дескриптори використовують для виведення/введення тексту?
7. Як можна визначити формат для багаторазового використання?
8. Напишіть програму обчислення об'єму циліндра, який визначається за формулою (1.7), та передбачте в цій програмі формування виведення текстових повідомлень, що мають спростити роботу користувача.

3 КЕРУВАННЯ ПОРЯДКОМ ВИКОНАННЯ ІНСТРУКЦІЙ ПРОГРАМИ

В мовах програмування передбачається можливість зміни прийнятої за замовчанням послідовності виконання інструкцій за їхнім розташуванням в тексті програми, що буває необхідним в деяких випадках. Робота та навіть форма запису інструкцій керування є подібними в різних мовах програмування, тому відноситься до загальних понять програмування.

3.1 Теоретичні відомості щодо керування порядком інструкцій

В деяких випадках необхідно змінювати прийняту за замовчанням послідовність виконання інструкцій за їхнім розташуванням в тексті програми.

3.1.1 Зміна послідовності виконання інструкцій програми здійснюється за допомогою спеціальних інструкцій переходу, що визначають, яку саме інструкцію слід виконувати із поточного місця програми. Для визначення інструкції програми, яку слід виконувати, в інструкціях переходу використовуються мітки, якими мають бути заздалегідь позначені необхідні інструкції програми. Таким чином, зміна послідовності виконання інструкцій програми може здійснюватися шляхом переходу до виконання лише інструкцій, що позначені мітками.

Безумовний перехід – це задана зміна послідовності виконання інструкцій програми незалежно від стану програми, що визначається сукупністю значень змінних.

Умовний перехід – це зміна послідовності виконання інструкцій програми відповідно до виконання деякої умови, яка визначається призначенням програми.

3.1.2 Реалізація безумовного переходу є найпростішим способом змінення послідовності виконання інструкцій програми та здійснюється за допомогою інструкції [3–5]:

GOTO label, (3.1)

де label – унікальна мітка, яка заздалегідь визначена, щоб помітити відповідну інструкцію програми, та являє собою цифри, які мають бути

розташованими у рядку відповідної інструкції з 1-го по п'ятий символи (старші нулі та пробіли ігноруються).

Інструкція (3.1) працює таким чином, що після її виконання програма виконує не ту інструкцію, що міститься безпосередньо за інструкцією (3.1), а інструкцію, що позначена міткою label. Після цього програма послідовно виконує інструкції, які розташовані слідом за інструкцією, що позначена міткою label. Слід зазначити, що стиль програмування, який використовує мітки та інструкції переходу, сьогодні вважається застарілим і не рекомендується для широкого використання. В той же час, навіть у найсучасніших мовах програмування передбачається можливість використання міток та безумовних переходів. Це обґрунтовується тим, що в деяких випадках використання міток та переходів є найпростішим засобом реалізації завдань програми. Тобто використання міток та переходів є цілком допустимим у сучасному програмуванні за умов, що кількість міток у програмі є невеликою.

3.1.3 Арифметична інструкція IF являє собою найпростішу реалізацію умовного галуження для зміни послідовності виконання частин програми відповідно до виконання чи невиконання певних умов. Арифметична інструкція IF має вигляд [3–5]:

$$\text{IF (ArExpr) label1, label2, label3,} \qquad (3.2)$$

де ArExpr – арифметичний вираз або ім'я змінної з числовим типом; label1, label2 та label3 – мітки, які позначають інструкції програми.

Інструкція (3.2) працює наступним чином: спочатку визначається результат розрахунків за арифметичним виразом ArExpr і далі здійснюється перехід до інструкції, що позначена міткою label1, або label2 або label3, якщо результат ArExpr арифметичного розрахунку виразу інструкції IF відповідно є від'ємним, дорівнює нулю, є додатним.

3.1.4 Більш широкі можливості порівняно із арифметичною інструкцією IF надає використання логічної інструкції IF, яка використовує логічний тип – особливий тип даних.

Логічний тип – це тип даних, які можуть приймати лише два значення: .TRUE. (так, істина), або .FALSE. (ні, неправда). Логічний вираз – це вираз, результат якого має логічний тип. Логічна інструкція IF має вигляд [3, 5]:

$$\text{IF (LogicExpr) Instruction ,} \qquad (3.3)$$

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

Інструкція (3.3) працює наступним чином: спочатку визначається результат за логічним виразом LogicExpr , і далі, якщо результат LogicExpr

дорівнює .TRUE., то виконується інструкція Instruction ; після цього програма продовжує виконання наступної інструкції. Зазначимо, що в якості Instruction може бути будь-яка інструкція, що виконується, у тому числі й інструкція безумовного переходу (3.1). Використання специфічного, незрозумілого для повсякденної свідомості, типу даних – логічного типу певним чином ускладнює початкове ознайомлення з інструкцією (3.3). Отже, для початкового ознайомлення наведемо приклад операції, яка має результатом логічний тип:

$$\text{val1.LE.val2}\,, \qquad (3.4)$$

де .LE. – набір символів, який в мові програмування FORTRAN символічно являє операцію відношення "менше або дорівнює" (англ. Less or Equal); val1 та val2 – арифметичні вирази, або імена змінних числового типу.

Неважко здогадатися, що інструкція (3.4) відповідає перевірці умови $\text{val1} \le \text{val2}$; у випадку, якщо ця умова виконується, то результатом інструкції логічного виразу (3.4) буде значення .TRUE., а у випадку невиконання цієї умови результатом буде значення .FALSE.

3.2 Завдання щодо керування виконанням інструкцій програм

Зрозуміло, що відповідно до фізичного змісту усі вихідні дані розрахунку за формулою (1.6), а саме – внутрішній тиск p, внутрішній діаметр D та допустиме напруження $[\sigma]$ матеріалу, а також результат розрахунків – товщини стінки S_R, мають бути додатними числами. В той же час, при введені вихідних даних користувач програми може помилково ввести від'ємне значення напруження, та (або) внутрішнього діаметра, та (або) допустимого напруження матеріалу, і в кожному із цих випадків результат розрахунку за формулою (1.6) не матиме змістовного сенсу. При помилковому введені вихідних даних результат розрахунків за формулою (1.6) може виявитися від'ємним, що покаже користувачу про помилки введення вихідних даних, але навіть при неправильно введених вихідних даних результат розрахунків може виявитися додатною величиною, і користувач не помітить похибок введення вихідних даних. З урахуванням цих обставин, щоб уникнути помилок в розрахунках, має сенс перевіряти правильність та коректність введення вихідних даних.

Слід розуміти, що не усі можливі додатні вихідні дані щодо внутрішнього тиску p, внутрішнього діаметра D та допустимого напруження $[\sigma]$ матеріалу є коректними для виконання розрахунку за формулою (1.6). Дійсно, за умов, коли внутрішній тиск p є занадто великим у порівнянні

із допустимим напруженням $[\sigma]$ матеріалу, результат розрахунку за формулою (1.6) буде некоректним, бо приведе до від'ємної товщини S_R стінки. Таким чином, введені вихідні дані щодо внутрішнього тиску та допустимого напруження мають задовольняти наступній умові:

$$2[\sigma] > p\,. \tag{3.5}$$

Отже, для перевірки коректності введення вихідних даних для розрахунку за формулою (1.6) слід перевірити, щоб усі введені дані були додатними числами та задовольняли при цьому умові (3.5).

3.3 Приклади керування виконанням інструкцій в програмах

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

3.3.1 Розглянемо спочатку використання арифметичної інструкції IF, що є зручно для перевірки знаку введених значень вихідних даних розрахунку за формулою (1.6).

Приклад програми, в якій виконується розрахунок за формулою (1.6) та виконується попередня перевірка введених вихідних даних із повідомленням користувача програми у випадку некоректного введення вихідних даних, наведений в ліст. 3.1. У наведеному тут прикладі арифметичні інструкції вигляду (3.2) були використані для перевірки коректного введення вихідних даних, які відповідно умовам розрахунку (1.6) мають бути додатними, а безумовний перехід вигляду (3.1) використано, щоб програма не виводила на екран монітора повідомлення про похибку введених вихідних даних у випадку, коли такої помилки немає. Зазначимо, що дві із трьох міток в арифметичних інструкціях IF збігаються у наведеному прикладі програми, що не забороняється мовою програмування.

Лістинг 3.1 – Використання арифметичної інструкції IF

```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT 100,'INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      IF (P) 210,210,220
220   PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      IF (D) 210,210,230
230   PRINT 100,'INPUT PERM STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
```

```
      IF (SIGMA) 210,210,240
240   SR=P*D/(2*SIGMA-P)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
      PRINT'(A,E15.8,1X,A)','SR=',SR,'(MM)'
      GOTO 300
210   PRINT*,'ERROR INPUTED DATA'
100   FORMAT (A$)
110   FORMAT (A,F6.2,1X,A)
300   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

3.3.2 Розглянемо спочатку використання логічної інструкції IF, що є зручно для перевірки знаку введених значень вихідних даних розрахунку за формулою (1.6).

Приклад програми, в якій виконується розрахунок за формулою (1.6) та виконується попередня перевірка введених вихідних даних із повідомленням користувача програми у випадку некоректного введення вихідних даних, наведений в ліст. 3.2. У наведеному прикладі (ліст. 3.2) логічні інструкції вигляду (3.3) використані разом із логічним виразом вигляду (3.4) для перевірки коректного введення вихідних даних, які відповідно умовам розрахунку (1.6) мають бути додатними. Безумовний перехід (ліст. 3.2) вигляду (3.1) використано, щоб програма не виводила на екран монітора повідомлення про похибку введених вихідних даних у випадку, коли такої помилки немає.

Лістинг 3.2 – Використання логічної інструкції IF

```
      PROGRAM SR
      REAL SR,P,D,SIGMA
      PRINT 100,'INPUT PRESSURE, P(MPA):'
      READ'(F6.2)',P
      IF (P.LE.0) GOTO 210
      PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      IF (D.LE.0) GOTO 210
      PRINT 100,'INPUT PERM STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      IF (SIGMA.LE.0) GOTO 210
      SR=P*D/(2*SIGMA-P)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
      PRINT'(A,E15.8,1X,A)','SR=',SR,'(MM)'
```

```
      GOTO 300
210   PRINT*,'ERROR INPUTED DATA'
100   FORMAT (A$)
110   FORMAT (A,F6.2,1X,A)
300   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
      END
```

Якщо порівняти програми, що наведені в ліст. 3.1 та ліст. 3.2, то можемо зробити висновок, що використання логічної інструкції IF потребує меншої кількості міток, ніж використання арифметичної інструкції IF, і це є більш зручним. Безумовно, що в мові програмування FORTRAN передбачена можливість різних операцій відношення, а не тільки операції типу (3.4), що буде розглянуто у подальшому.

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. Яким чином здійснюється зміна послідовності виконання інструкцій програми?
2. Що таке мітки, як їх оформлюють та для чого вони потрібні?
3. Що таке безумовний перехід і за допомогою якої інструкції він реалізується в програмах?
4. Що таке умовний перехід і за допомогою якої інструкції він реалізується в програмах з використанням арифметичних даних?
5. Що таке дані логічного типу та яким чином умовний перехід реалізується в програмах з використанням даних такого типу?
6. Яким чином в програмі можна порівняти два значення?
7. Напишіть програму обчислення об'єму циліндра, який визначається за формулою (1.7), та передбачте в цій програмі перевірку коректності введених даних.

4 РОЗШИРЕНІ МОЖЛИВОСТІ КЕРУВАННЯ ПОРЯДКОМ ВИКОНАННЯ ІНСТРУКЦІЙ

Існуючі мови програмування надають широких можливостей щодо керування порядком виконання інструкцій в програмах, але реалізація таких можливостей в різних мовах програмування здійснюється схожим чином. Керування порядком виконання інструкцій досить широко використовується в програмах різноманітного призначення, тому відповідні питання відносяться до загальних понять програмування.

4.1 Теоретичні відомості про розширені можливості керування

В деяких випадках необхідно змінювати відповідно поточному стану програми та досить складним умовам досить прийняту за замовчанням послідовність виконання інструкцій за їхнім розташуванням в тексті програми.

4.1.1 Розширені можливості щодо керування порядком виконання інструкцій програм засновані на використанні особливого типу даних – логічного типу [3, 5].

Логічний тип даних (LOGICAL) – це тип даних, які можуть приймати лише два значення: .TRUE. (так, істина), або .FALSE. (ні, неправда). Логічний тип можна уявляти як два можливі варіанти відповіді на питання щодо правдивості деякого твердження, яке може бути або істинним (правдивим), або ні.

Логічний тип використовується для перевірок, чи задовольняють значення змінних програми певним умовам. Для цього в мовах програмування передбачаються операції відношення між значеннями змінних та (або) констант. Такі операції відношення є символічною формою (кодуванням) представлення умов, яким мають задовольняти два значення:

$$\text{val1 relop val2}, \qquad (4.1)$$

де relop – набір символів, який символічно представляє операцію відношення; val1 та val2 – арифметичні вирази, або імена змінних числового типу.

Результатом операції відношення (4.1) є логічний тип, який набуває значення .TRUE. (так, істина), якщо відношення (4.1) виконується, та значення .FALSE. (ні, неправда), якщо відношення (4.1) не виконується; зрозуміло, що порівнюватися мають значення, які принципово можна порівнювати. В якості val1 та val2 замість імен змінних та констант може виступати арифметичний вираз, що містить імена змінних та константи. В мові програмування FORTRAN передбачено шість операцій порівняння, які наведені в табл. 4.1.

Таблиця 4.1 – Операції відношення в мові програмування FORTRAN

Символьний запис операції	Змістовний сенс операції
.LT.	менше
.LE.	менше або дорівнює
.EQ.	дорівнює
.NE.	не дорівнює
.GE.	більше або дорівнює
.GT.	більше

Більш складні, ніж наведені у табл. 4.1, умови можна формулювати із використанням логічних операцій – операцій над величинами логічного типу, результат яких також є логічним типом. Відомо п’ять основних логічних операцій, які наведені в табл. 4.2

Таблиця 4.2 – Логічні операції

Символьний запис операції	Змістовний сенс операції
.NOT.	Логічне заперечення (ні)
.AND.	Кон’юнкція (і)
.OR.	Диз’юнкція (або)
.EQV.	Логічна еквівалентність
.NEQV.	Логічна нееквівалентність

В якості аргументів логічних операцій (табл. 4.2) можуть виступати дані логічного типу та операції відношення (табл. 4.1).

4.1.2 Найбільш широкими можливостями щодо умовного керування послідовністю виконання інструкцій програми є структурна інструкція IF, яка дозволяє керувати виконанням послідовностей інструкцій у залежності від задоволення певної заданої умови. Така інструкція в тому чи іншому вигляді присутня в різних мовах програмування, а в FORTRAN має наступний вигляд [3, 5]:

```
IF (cond) THEN
instr for if
ELSE
instr for else
END IF
```
(4.2)

Тут cond – позначено логічну змінну, або вираз, якими визначають необхідну умову; instr for if позначено інструкції, які мають бути виконані за умов, коли значення cond дорівнює .TRUE., а instr for else по-значено інструкції, які мають виконуватися у протилежному випадку. Передбачено, що інструкція ELSE може бути відсутньою. Структурна форма інструкції керування дозволяє забезпечити виконування не одного, як в логічній інструкції IF, а декількох інструкцій при виконанні заданої певної умови. Слід підкреслити, що використання конструкцій вигляду IF–THEN–ELSE є характерним та здійснюється однаково в багатьох універсальних мовах програмування, таких як PASCAL, C, Python та багатьох інших.

4.2 Приклади використання розширених можливостей керування

Покажемо використання розширених можливостей керування порядком виконання інструкцій програм.

4.2.1 Прикладом використання в простій програмі змінних логічного типу та деяких із операцій відношення (табл. 4.1), може виступати повідомлення користувача про характер введеного числового значення:

Лістинг 4.1 – Використання логічної змінної

```
      PROGRAM PR4A
      REAL A
      LOGICAL COND
      PRINT 100,'INPUT VALUE, A:'
      READ'(F6.2)',A
      IF (A.LT.0) PRINT*,'A<0'
      COND=A.EQ.0
      IF (COND) PRINT*,'A=0'
      IF (A.GT.0) PRINT*,'A>0'
      PAUSE 'PRESS ENTER TO EXIT PROGRAM'
100   FORMAT (A$)
      END
```

Наведена в ліст. 4.1 програма повідомляє користувача про знак введеного ним числового значення дійсної змінної.

4.2.2 Прикладом використання деяких логічних операцій (табл. 4.1) може виступати повідомлення користувача про характер введеного числового значення відносно деякого інтервалу, як це зроблено в програмі, текст якої наведений в ліст. 4.2. Наведена програма (ліст. 4.2) пропонує користувачу ввести значення $A_{\min}$ та $A_{\max}$, повідомляє про помилку, якщо $A_{\min} \geq A_{\max}$, а в іншому випадку пропонує ввести значення A і повідомляє користувача про те, чи виконуються умови $A < A_{\min}$, $A > A_{\max}$ та $A_{\min} \leq A \leq A_{\max}$.

4.2.3 Використання структурної форми інструкції керування дозволяє зменшити кількість міток та переходів у програмі та завдяки цьому значно спрощує розуміння тексту програми, що можна побачити на прикладі програми, що виконує розрахунок за формулою (1.6), текст якої показано ліст. 4.3.

Лістинг 4.2 – Використання логічних операцій

```
      PROGRAM PR4B
      REAL A,AMIN,AMAX
      LOGICAL COND
      PRINT '(A$)','INPUT VALUE, AMIN:'
      READ'(F6.2)',AMIN
      PRINT '(A$)','INPUT VALUE, AMAX:'
      IF (AMIN.GE.AMAX) GOTO 200
      READ'(F6.2)',AMAX
      PRINT '(A$)','INPUT VALUE, A:'
```

```
      READ’(F6.2)’,A
      IF (A.LT.AMIN) PRINT*,’A<AMIN’
      IF (A.GT.AMIN) PRINT*,’A>AMIN’
      COND=(A.GE.AMIN).AND.(A.LE.AMAX)
      IF (COND) PRINT*,’AMIN<=A<=AMAX’
      GOTO 300
200   PRINT*,’ERROR AMIN>=AMAX’
300   PAUSE ‘PRESS ENTER TO EXIT PROGRAM’
      END
```

В програмі, що наведено в ліст. 4.3, здійснюється розрахунок товщини стінки циліндричного корпусу ядерного реактора за формулою (1.6) після перевірок правильного введення вихідних даних розрахунку. Використання структурної інструкції керування дозволило істотно зменшити кількість міток та переходів в програмі, хоча змістовна наповненість програми стала значно розширеною за рахунок формування точних повідомлень про характер помилок при введенні вихідних даних. Такі структурні інструкції керування із використанням логічного типу даних дозволяють керувати виконанням достатньо великої кількості послідовностей інструкцій за виконанням достатньо великої кількості достатньо складних умов, які можуть знадобитися при розробці програм різного призначення.

4.3 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 дані логічного типу та як визначаються змінні логічного типу?
2. Які операції відношення передбачені в FORTRAN?
3. Які логічні операції передбачені в FORTRAN?
4. Який вигляд має структурна форма запису інструкції для керування порядком виконання інструкцій програми?
5. Напишіть програму обчислення об’єму циліндра, який визначається за формулою (1.7), та передбачте в цій програмі перевірку коректності введених даних із виведенням текстових повідомлень про характер похибок.

Лістинг 4.3 – Використання структурної інструкції керування

```
      PROGRAM SR4
      REAL SR,P,D,SIGMA
      PRINT 100,’INPUT PRESSURE, P(MPA):’
      READ’(F6.2)’,P
```

```
      IF (P.LE.0) THEN
      PRINT*,'ERROR INPUTED DATA, P<=0'
      GOTO 200
      ENDIF
      PRINT 100,'INPUT DIAMETER, D(MM):'
      READ'(F6.2)',D
      IF (D.LE.0) THEN
      PRINT*,'ERROR INPUTED DATA, D<=0'
      GOTO 200
      ENDIF
      PRINT 100,'INPUT PERM STRESS, SIGMA(MPA):'
      READ '(F6.2)',SIGMA
      IF (SIGMA.LE.0) THEN
      PRINT*,'ERROR INPUTED DATA, SIGMA<=0'
      GOTO 200
      ENDIF
      SR=P*D/(2*SIGMA-P)
      PRINT 110,'P=',P,'(MPA)'
      PRINT 110,'D=',D,'(MM)'
      PRINT 110,'SIGMA=',SIGMA,'(MPA)'
      PRINT 110,'SR=',SR,'(MM)'
200   PAUSE 'PRESS ENTER TO EXIT PROGRAM'
100   FORMAT (A$)
110   FORMAT (A,F6.2,1X,A)
      END
```

ПЕРЕЛІК ПОСИЛАНЬ

1. Chapman S.J. Fortran for Scientists and Engineers / S.J. Chapman. – New York: McGrawHill Education, 2018. – 1024 p.

2. Markus A. Modern Fortran in Practice / A. Markus. – New York: Cambridge University Pres, 2012. – 253 p.

3. Brauch W. Programmierung mit FORTRAN : Eine Einführung in die Datenverarbeitung und Basic FORTRAN IV / W. Brauch. – Stuttgart: Teubner Verlag, 1981. – 224 s.

4. Фортран: Программированное учебное пособие / Под ред. чл.-кор. АН УССР Е. Л. Ющенко. – Киев: Вища школа, 1980. – 400 с.

5. Nyhoff L. FORTRAN 77 and Numerical Methods for Engineers and Scientists / L. Nyhoff, S. Leestma. – London: Macmillan Pub Co, 1995. – 700 p.

6. http://openwatcom.org

7. Реактори і парогенератори енергоблоків АЕС: схеми, процеси, матеріали, конструкції, моделі / О. В. Єфімов, М. М. Пилипенко, Т. В. Потаніна та ін. ; за ред. О. В. Єфімова. – Харків : ТОВ «В справі», 2017. – 420 с.