Давидова А. В.

КОНСПЕКТ ЛЕКЦІЙ

З КУРСУ

ОСНОВИ ПРОГРАМУВАННЯ

Дніпро 2022

Метою даного курса є вивчення певної мови програмування, оволодіння засобами сучасних інформаційних та комунікаційних технологій в обсязі, достатньому для навчання та професійної діяльності.

Слухачі мають можливість ознайомитися з логічною будовою комп’ютерів, навчити прийомам обчислення з використанням електронних таблиць та основам програмування..

У результаті вивчення навчальної дисципліни студент повинен вміти пояснювати свої рішення і підґрунтя їх прийняття фахівцям і неспеціалістам в ясній і однозначній формі.

Для більш якісного опанування предмету здобувач вищої освіти вищого навчального закладу має знати основні поняття алгоритму та мови програмування, структурний підхід до програмування, синтаксис мови програмування FORTRAN у стандартному визначенні та впевнено вміти користуватися одним з інтегрованих середовищ розробника FORTRAN, розробляти та налагоджувати прості програми на мові FORTRAN.

Організація даних

Текст Фортран-програми можна записати у фіксованому форматі (з твердим закріпленням позицій за мітками, операторами і коментарями) чи у вільному форматі. Формат відіграє роль тільки при компіляції і не впливає на наступні етапи технологічного циклу (компілятор за замовчуванням вважає програми, записані у файлах з розширенням.f90, що мають вільний формат, а з .f і .for - фіксований). Фіксований формат застосовується для сумісності зі старими версіями мови. Тут позиції 1-5 виділяються для запису мітки, а 7-72 - для тексту єдиного оператора. Відмінний від нуля і пробілу символ у шостій позиції вказує, що рядок продовжує раніше початий оператор (продовження може мати до 19 рядків. Знак оклику в будь-якій позиції, крім шостий, служить початком коментарю.

У вільному форматі довжина рядка допускається до 132 символів і рпактически визначається шириною рядка пристрою висновку. Оператор може мати до 54 рядків продовження. Ознакою наявності продовження є завершальний рядок амперсанд &. При записі нескольних коротких операторів підряд їх можна розмістити в одному рядку, розділивши крапкою з коми.

Типи даних розділяються на вбудовані і похідні, створювані користувачем.

Вбудовані типи даних

INTEGER	цілий
REAL	речовинний або реальний
COMPLEX	комплексний
LOGICAL	логічний
CHARACTER	символьний

Символьний тип даних можуть мати змінні і константи, що ми будемо називати рядками, а також масиви і функції. Елементом символьного масиву є рядок. Результат, що повертається символьною функцією, також є рядком.

Оператором присвоювання в перемінну символьного типу встановлюється результат символьного вираження:

символьна перемінна = символьний вираз.

Фортран містить єдину символьну операцію - операцію ***конкатенації***, що позначається (//). Результатом операції є об'єднання рядків - операндів символьного вираження. Довжина результуючої рядка дорівнює сумі довжин строк-операндів.

Приклад опису змінної символьного типу

```
Character st*20, im*10, fam*10
Im = 'Ivan'
Fam = 'Petrenko'
St = im//fam.
```

Оператор присвоювання позначається знаком рівності (=) і записується у вигляді varname = вираз.

У результаті присвоювання змінна varname одержує нове значення, що повертається в результаті обчислення вираження. При цьому необхідно враховувати, що тип змінної varname повинний збігатися з типом виразу.

Оператор DATA дає можливість присвоювати змінним початкові значення. Загальний вигляд оператору:

DATA *список імен змінних /список значень змінних/*

Приклади:

```
DIMENSION M1(4), M2(3:5), K1,K2(2,3), M3(4)
……
DATA A/1.2/ B,C/3.5,4.0/
DATA M1/10,20,30,40/
DATA M2/1,2,3/
DATA K1/10,20,30,40,50,60/
DATA ((K2(I,J), J=1,2),I=1,3)/10,30,50,20,40,60/
DATA M3/4*1/
```

Для введення інформації з клавіатури застосовується оператор **Read,** а для виводу на екран застосовуються оператори **Print** та **Write**.

Оператор PRINT – здійснює вивід на друк.

Безформатний вивід: **PRINT *, *список змінних.***

Вивід за форматом: **PRINT M1, *список змінних,*** де **M1** - номер мітки, яка стоїть біля відповідного оператора **FORMAT.**

Оператор FORMAT дає можливість керувати розміщенням інформації, яка виводиться на друк. Загальний вигляд оператора:

FORMAT (*список специфікаторів*)

Специфікатори у списку повинні відповідати типам змінних, які розміщені у списку змінних оператора **PRINT.** Після вичерпання списку специфікаторів , якщо список змінних ще не закінчився, відбувається повернення до першого специфікатора у списку.

У разі потреби перед окремими специфікаторами або групами специфікаторів, взятих у дужки, може стояти так званий повторювач – ціле число, яке означає кількість повторень даного специфікатора або даної групи специфікаторів.

Список вводу - частина оператора вводу, що установлює величини, які потрібно ввести. Відповідно, список виводу установлює величини, які потрібно вивести.

Приведемо приклади основних видів специфікацій формату:

i 10 - ціле, поле з 10 позицій,

f 10.5 - речовинне, поле 10 позицій, 5 знаків після крапки,
e 12.4 - речовинне з крапкою, що плаває, у поле з 12 позицій; мантиса з 4 цифр зі старшою цифрою відразу після крапки; показова частина складається з букв e, знака і двох цифр.

На оператор формату можливі багаторазові посилання. Звичайно оператори формату поміщають безпосередньо за їхніми операторами обміну, що використовують, чи збирають в одному місці програмної одиниці.

Перехід на новий запис (рядок) у специфікації формату задається слэшем (/). Подвійний слэш забезпечить пропуск чистого рядка. Зворотний слэш указує необхідність продовження колишнього запису (цей засіб часто використовується при висновку на термінал).

Основні специфікатори:

Формат	Тип даних	Примітки
In	Цілі числа	n-число позицій
Fn.m	Реальні числа	n-число позицій, m-число десятичних знаків
En.m	Реальні числа в експотенційній формі	n-число позицій, m-число десятичних знаків
Dn.m	Реальні числа подвійної точності	n-число позицій, m-число десятичних знаків
'…'	Розміщення коментарів	
nX	n пробілів	
/	Перехід на нову строку	

Приклад:

```
DIMENSION F(4,5)
……
PRINT 2, A,B,I1,D
PRINT 4,((F(I,J),J=1,5),I=1,4)
2  FORMAT(3X, 2F6.3, I3)
4  FORMAT(3X,'МАСИВ F',/,4(10X,5F5.2,/))
```

Якщо до тексту програми треба ввести якісь пояснення, то рядки, де вони розміщені, помічаються у першій позиції символом C або *. Такі рядки сприймаються як коментарі і пропускаються при виконанні програми.

Арифметичні вирази та функції

Арифметичні операції розрізняються пріорітетом:

** - операція зведення в ступінь (операція з найвищим пріорітетом);
* - операція множення;
/ - операція ділення;
+ - операція додавання;
– - операція віднімання.

Стандартні математичні функції:

Ім'я	***Математичний вигляд***	***Примітки***
SQRT(X)	$\sqrt{x}$	Аргумент більше 0
EXP(X)	e^x	
ALOG(X)	ln x	Аргумент більше 0
ALOG10(X)	lg x	Аргумент більше 0
SIN(X)	sin x	Аргумент у радіанах
COS(X)	cos x	Аргумент у радіанах
TAN(X)	tg x	Аргумент у радіанах
ыSIN(X)	arcsin x	Аргумент менше або дорівнює 1
ACOS(X)	arccos x	Аргумент менше або дорівнює 1
ATAN(X)	arctg x	
SINH(X)	sh x	
COSH(X)	ch x	
TANH(X)	th x	
ABS(X)	|x|	

Оператор-функція дає можливість користувачу самому задати функцію, яку потім можна використати так само, як і стандартну. Загальний вигляд оператору:

$$\mathbf{FUN(P_1,P_2,P_N)=W,}$$

де **FUN –** ім'я функції;
P_1, P_2, P_N – аргументи;
W – арифметичний вираз.

Ідентифікатор імені функції **FUN** повинен мати той самий тип, що і арифметичний вираз **W**.

Кількість фактичних параметрів при звернені до оператора – функції повинна дорівнювати кількості формальних параметрів в описі функції, а тип кожного формального параметру повинен співпадати з типом відповідного фактичного параметру.

Оператори управління

У нормальному режимі оператори головної програми чи процедури виконуються в порядку їхнього запису. Для зміни цього порядку служать *оператори управління. Управляюча* умова завжди є скаляром чи приводиться до скаляра.

Деякі оператори управління мають *блокову* конструкцію, яка починається з ключового оператора-заголовка, може містити проміжні ключові оператори та закінчується парним заголовку оператором завершення. Ввійти в таку конструкцію можна тільки через її заголовок. З блоку можливі переходи

- На інший оператор того ж блоку;
- На заключний оператор тієї ж конструкції;
- На оператор, що знаходиться поза даною конструкцією.

Конструкції, що виконуються, можуть бути вкладеними.

Оператор безумовного переходу – **GOTO (**або **GO TO m,** де **m –** мітка оператору до якого здійснюється перехід). При складанні програми ператор безумовного *переходу* має вид

go to <мітка>

Мітка (числова) повинна бути при *виконуваному* операторі.

Окремої уваги заслуговує *призначуваний* перехід. У цьому випадку в програму включається оператор виду

go to <ціла_змінна> (<список_міток>)

Сучасний стиль рекомендує "програмування без go to ", і FPS має достатній набір обговорюваних нижче альтернатив оператором переходу. Однак можливі ситуації, коли go to виявиться найбільш простим і логічним засобом.

Логічний умовний оператор має вид

IF (w) t,

де **w** – логічний вираз (умова), **t** – виконуваний оператор. Якщо значення логічного виразу – істина, то виконується оператор **t,** якщо ні – оператор **t** пропускається.

При складанні програми логічний умовний оператор має вид

If(<скалярне ЛВ>) <дія>

Тут і далі ЛВ- логічне вираження. Воно повинно записуватися обов'язково в дужках. Приклад:

If (x<y) z=x.

Якщо необхідно зв'язати з умовою виконання декількох дій, застосовується більш складна блокова конструкція, яка має назву блочний умовний оператор.

Блочний умовний оператор при складанні програми має вид

IF (w_0)THEN
блок$_0$
ELSE IF (w_1) THEN
блок$_1$
ELSE IF (w_2) THEN
блок$_2$
.....
ELSE IF (w_n) THEN
блок$_n$
ELSE
блок
END IF

де $\mathbf{w_i}$ – логічний вираз (умова)

Кожний із блоків може містити довільну послідовність виконуваних операторів. При відсутності альтернативи другий блок разом з else опускається. Нарешті, при необхідності в додаткових перевірках використовується else if.

Арифметичний умовний оператор має вид

IF (w) $\mathbf{m_1, m_2, m_3}$,

де **w –** арифметичний вираз $\mathbf{m_1, m_2, m_3}$ – мітки.

Якщо **w<0,** здійснюється перехід до оператора з міткою $\mathbf{m_1}$, при **w=0** або- до оператора з міткою $\mathbf{m_2}$, при **w>0** – до оператора з міткою $\mathbf{m_3}$.

Логічні операції

.AND. – логічне і (кон'юнкція)	**.XOR.** - логічне або (що виключає)
.OR. – логічне або (діз'юнкція)	**.EQV.** -еквівалентність
.NOT. - логічне ні (заперечення)	**.NEQV.** - не еквівалентність

Операції відношень можуть бути записані в двох формах:

.LT. - або < (менше)	**.NE.** - або ≠ (не дорівнює)
.**LE.** - або ≤ **(** менше чи дорівнює)	**.GT.**- або > (більше)
.EQ.- або == (дорівнює)	**.GE.** - або ≥ (більше чи дорівнює)

Пробіли при запису позначення операції є помилкою.
Приклад: **a.le.b** - вірний запис
b < c - не вірний запис

Арифметичний оператор циклу

Однотипні обчислення, що йдуть підряд, можна запрограмувати однократно за допомогою оператора переходу на початок відповідної групи команд ("тіла циклу") - зрозуміло, після його настроювання на чергове повторення. Однак ці функції у всіх мови програмування реалізуються більш наочними *операторами циклу*. Усі цикли у Фортрані будуються з *верхнім закінченням*, тобто необхідність дій з тіла циклу перевіряється перед першим виконанням вхідних у нього операторів.

Арифметичний оператор циклу має вигляд

DO M, I=I1, I2, I3
блок операторів
M *виконуваний оператор або* CONTINUE

де **M –** мітка, що розміщується біля останнього оператору циклу;
I – параметр циклу; **I1 –** початкове значення параметру циклу; **I2 –** кінцеве значення параметру циклу; **I3 –** шаг циклу. У разі, коли **I3** відсутнє, шаг дорівнює 1.

Шаг може бути і негативним. Усі параметри можуть бути константами чи виразами зі стандартних числових типів (integer, real, double precision), змінна циклу також належить до того ж типу. Параметри і необхідна кратність виконання обчислюються до входу в цикл.

Змінна циклу не повинна перевизначатися в тілі циклу (ця помилка виявляється компілятором). Якщо змінна циклу – типу real, погрішності представлення поточного і граничного її значень можуть привести до помилки в числі повторень.

Масивом називається упорядкована безліч однотипних значень (список, проекції вектора, таблиця результатів експерименту, матриця перетворення координат і т.д.). Елементи масивів звичайно обробляються деяким стандартним образом. Посилання на елемент масиву складаються з загального імені масиву та набору індексів (у круглих дужках), які характеризують положення в ньому обраного елемента, тобто A(i,j) означає елемент матриці A на перетинанні i-того рядка та j-го стовпця. Тому відпадає потреба в сотнях різних імен для родинних об'єктів програми. Два масиви вважаються рівними, якщо рівні їхні елементи, що знаходяться в однакових позиціях.

Індекси самі можуть бути елементами масивів (така можливість у попередніх версіях Фортрану була відсутня).

Програми рішення багатьох задач вимагають декількох циклів, наприклад при упорядкуванні масивів. У цьому випадку доцільно використовувати вкладені цикли. Для цього дуже важливо правильно визначити структуру майбутньої програми - насамперед кількість і відносне розташування циклів. Наприклад, суму всіх елементів масиву можна порахувати в такий спосіб:

```
Dimension R(5,6)
……….
do i=1,5
    s(i)=0
    do  j=1,6
      s(i)=s(i)+r(i,j)
      sum=sum+r(i,j)
end do
     end do
        end
```

Аналіз даного приклада дозволяє сформулювати дві рекомендації із програмування вкладених циклів:

- Внутрішні цикли необхідно приводити у вихідний стан - це стосується операцій нагромадження різного виду - безпосередньо перед його початком (зверніть увагу на положення оператора обнуління s);
- Внутрішні цикли, що визначають основну трудомісткість виконання програми потрібно будувати дуже ощадливо, виносячи з них нагору чи вниз по програмі повторювані обчислення й операції індексування.

Вкладеність циклів для наочності потрібно указувати відступами. При глибокій вкладеності слід робити заголовок і кінець циклу іменованими. Корисно також побудова циклів з числовими мітками типу:

```
        do 10  i=1,30
            s=s+x(i)
10     end do
```

Тіло циклу можна обмежити й іншим позначеним виконуваним оператором. На вигляд заключного оператора накладаються деякі обмеження, тому цикл із числовою міткою рекомендується закінчувати позначеним оператором продовження continue, який не виконує ніяких дій.

Достроковий (по додатковій умові) вихід з циклу на наступний за end do оператор можливий за допомогою оператора:

exit [< ім'я >].

При заданому імені вихід виробляється з позначеного цим им'ям циклу. За замовчуванням вихід буде тільки із самого внутрішнього циклу, що безпосередньо охоплює exit. Аналогічно оператор

cycle [< ім'я >]

виводить на керуючу конструкцію поіменованного циклу.

Операції над масивами

Для виконання відповідних дій з масивами використовуються різноманітні оператори. Вони поділяються на оператори пошуку в масивах та оператори які виконують певні дії з масивами.

1. Оператори пошуку в масивах.

Для знаходження максимального елемента довільного масиву використовують оператор maxval, а для знаходження мінімального елемента довільного масиву використовують оператор minval/

Для знаходження індексів максимального елемента довільного масиву використовують оператор maxljc, а для знаходження індексів мінімального елемента довільного масиву використовують оператор minloc.

2. Оператори, які виконують певні дії з масивами.

Для знаходження суми всіх елементів довільного масиву використовують оператор sum, а для знаходження добутку всіх елементів довільного масиву використовують оператор product.

Приклади використання операторів. Якщо задан одномірний массив A(5), який містить 5 довільних елементів, то суму всіх елементів цього масиву знаходять за допомогою оператора sum(A). При цьому немає необхідності будь-якій змінній присвоювати це значення. Відповідну дію можно виконати за допомогою оператора print. Print *, sum(A). Конструкція типу w= sum(A) необов’язкова.

4. Знайти максимальний елемент масиву $\{a_{i,j}\}$. На друк вивести значення знайденого елемента та його номер у вигляді виразу без формата.

5. Знайти суму та добуток всіх елементів масиву $\{a_{i,j}\}$.На друк вивести значення знайденого елемента зі знаком рівності без формата.

6. Підрахувати кількість від’ ємпних елементів масиву $\{a_{i,j}\}$ та знайти суму їх квадратів.На друк вивести значення знайдених елементів зі знаком рівності без формата.

7. Знайти мінімальні елементи масивів $\{b_i\}$ та $\{c_j\}$. На друк вивести значення мінімальних елементів та їхні номери у вигляді виразу без формата.

8. Виконати необхідні обчислення, згідно з варіант

Оператор циклу з умовою

Оператор циклу з умовою має вигляд:

DO M WHILE (w)
блок операторів
M *виконуваний оператор* або CONTINUE
або
DO WHILE (w)
блок операторів
END DO

де **M –** мітка, що розміщується біля останнього оператору циклу; **w –** логічний вираз.

Цикл виконується до тої пори, доки значення логічного виразу дорівнює "*істина*". Звичайно цикли з умовою використовуються при заздалегідь невідомій кількості повторень.

Перерахунок членів ряду можна робити не тільки від попереднього до поточного, але і від поточного до наступного. Правило перерахування і порядок проходження операторів у тілі циклу впливають на початкові значення, що задаються перед входом у цикл, змінних.

При роботі з циклами " **до тої пори** " і "**до**" треба стежити, щоб логічне вираження чи рано пізно прийняло значення *неправда* .Інакше відбудеться зациклення - "*нескінченне*" виконання операторів циклу.

Оператор переходу по ключу

Оператор переходу по ключу має такий вигляд

SELECT CASE (вираз)

CASE (значення 1)

блок операторів 1

CASE (значення 2)

блок операторів 2

……..

CASE (значення n)

блок операторів n

CASE DEFAULT

(значення)

END SELECT

При виконанні оператору SELECT CASE управління передається блоку операторів, для якого значення виразу співпадає із значенням, яке вказане у відповідному операторі CASE. У випадку, коли значення не співпадає із вказаним управління передається блоку оператора CASE DEFAULT.

Зовнішні функції та підпрограми

Програма на Фортрані являє собою певним чином оформлену сукупність операторів. Вона може бути монолітною чи складовою.

Мінімальний склад закінченої програми – тільки *головна програма*. Однак будь-яка досить складна задача вимагає її розгляду на декількох рівнях детальності, що визначає ієрархічну побудову програми з декількох субпрограмм (процедур). Звичайно з'ясовується, що деякі з приватних задач відносяться до стандартного арсеналу і тому відносяться до типових, існуючих для багаторазового застосування. Використання процедур робить програму в цілому компактною і доступною для огляду, зменшує трудомісткість розробки й особливо налагодження, дозволяє розділити розробку між декількома виконавцями, полегшує внесення виправлень у складний алгоритм з однотипними фрагментами.

Процедури можуть бути внутрішніми, зовнішніми і модульними. Опис *внутрішньої* процедури безпосередньо включається в текст програмної одиниці, (носія) і не може містити вкладених процедур. *Зовнішня* процедура існує автономно і може бути розроблена на інших мовах (*C* та його версії, ассемблер).

Програмні одиниці бувають наступних видів: головна програма, зовнішня процедура, модуль і блок даних. Кожна з них

• фізично відділена від інших;

• починається оператором-заголовком, що містить специфічне для даної програмної одиниці ключове слово: program, subroutine, function, module, block data;

• закінчується пропозицією end з повторенням (іноді не обов'язковим, але що рекомендується) того ж ключового слова і власного імені програмної одиниці;

• обробляється компілятором окремо від інших.

Звертання до *функції* може здійснюватися безпосередньо із виразу, і повернення значення, використовується у тім же виразі. Це значення являє єдиний результат роботи функції (можливо, структурований, тобто масив). Він зіставляється імені функції; тому в заголовку функції звичайно відсутній відповідний параметр, а в специфікації для імені функції вказуються атрибути результату:

integer function sumints (x)

(при відсутності такого опису в заголовку чи внутрішніх операторах оголошення атрибутів тип результату буде призначений за замовчуванням відповідно до першої букви імені функції). Обчислення функції повинне закінчуватися присвоюванням її імені відповідного значення, після якого випливає завершальний оператор

end function [<ім'я>].

Ім'я функції повинне бути оголошене в процедурі, яка викликається з повернеными атрибутами значення.

Покажчик функції, викликаної з порожнім списком аргументів, повинний містити дужки.

При використанні зовнішньої процедури-функції програма може мати вид:

```
program pf
 real f, y, x
 y = (f (x) +2*f (2*x)) / (x - f (x+.5)**2)
 print *, 'y = ', y
end program pf

function f (a)
 real a
 f = sin (a) – eps(- a)
 end function f
```

Звертання до функції не повинне перевизначати змінних, використовуваних в одному операторі з її викликом – зокрема, її аргументів (вимога відсутності побічного ефекту). Обов'язковий оператор end обмежує текст програмного компонента і завершує її виконання. Припинення програми задається також оператором stop. Останній може супроводжуватися кодом зупинки – текстовою чи константою послідовністю цифр (до 5 знаків), що виводяться при спрацьовуванні даної зупинки. Цей прийом корисний при налагодженні і відпрацьовуванні аварійних ситуацій.

Зовнішня процедура відрізняється від головної програми тільки заголовком і ключовим словом у завершальній рядку. Вона приводиться в дію оператором call із викликаної програмної чи одиниці покажчиком функції.

Внутрішні процедури виглядають також, як і модульні, але не можуть мати вкладених у них процедур.

Після завантаження всього програмного комплексу в оперативну пам'ять керування передається головній програмі.

Заголовок процедури-підпрограми має вид:
SUBROUTINE<ім'я_процедури>(<список_формальних_параметрів>).

Формальні параметри призначені для опису стандартних дій, виконуваних процедурою. Виклик підпрограми здійснюється оператором
call<ім'я_підпрограми>(<список_аргументів>)
і обробляється в першому наближенні як її тіло (опис) із заміною формальних параметрів фактичними – аргументами. Звідси випливає вимога відповідності аргументів параметрам по кількості, типу і змісту. Аргумент, який заміщує вхідний параметр, може бути виразом; тоді його значення обчислюється і фіксується при вході в процедуру. Вихідному і переобумовленому параметрам може відповідати тільки мінна (зокрема, масив).

Ім’я спільної області пам’яті може бути відсутнє, у цьому разі змінні, які входять до списку утворюють неіменовані області пам’яті.

Метою використання спільних областей пам’яті є забезпечення можливості звернення до одного й того ж фрагмента мам’яті ЕОМ з різних модулів програми.