Лекція №11 ПОіП « НАЛАГОДЖЕННЯ ПРОГРАМ»
(Модуль 2 -)

План лекції.

Налагодження програм

Налагодження — це один з найскладніших етапів технологічного процесу розробки програм. На налагодження припадає біля 50 % трудомісткості із загальних витрат на створення комплексів програм. Під налагодженням розуміють процес, який дозволяє отримати програму, що функціонує з потрібними характеристиками в заданій сфері зміни вхідних даних. Таким чином, внаслідок налагодження програма має відповідати сукупності правил і показників якості, узятій за еталонну для даної програми.

Щоб полегшити налагодження і скоротити час його проведення, необхідно дотримуватися таких технологічних етапів:

- Розробка загального плану проведення налагодження, загальної методики перевірки складеної програми, а також системи необхідних для налагодження контрольних прикладів.
- Перевірка розроблених алгоритмів, вибір відлагоджувальних засобів, а також визначення місць, що контролюються, ділянок, величин.
- Реалізація плану налагодження для отримання на ПК тестових результатів, необхідних під час локалізації помилок; підготовка еталонних результатів для тестів.
- Введення, друкування і звірка тексту програми.
- Отримання за допомогою транслятора допоміжних таблиць і перевірка їх.
- Ретельний контроль перших результатів, що отримуються за новою програмою.

Процес налагодження програм містить (рис. 6.2):

- створення сукупності тестових еталонних значень і правил, яким має відповідати програма за функціями, що виконуються, структурі, правилам опису, значенням початкових і відповідних їм результативних даних;
- статичне тестування текстів розроблених програм і даних на виконання всіх заданих правил побудови і опису без виконання об’єктного коду;
- тестування програми з її виконанням в об’єктному коді з різними рівнями деталізації: детерміноване, стохастичне і тестування в реальному масштабі часу;

- діагностику і локалізацію причин відхилення результатів тестування від заданих еталонних значень і правил;
- розробку зміни програми з метою виключення причин відхилення результатів від еталонних;
- реалізацію коригування програми, що забезпечує відповідність програми заданому еталону.

Рис. 6.2. Категорії тестів по об’єктах тестування

Основні групи помилок

Існує багато помилок, що допускають програмісти, точно так само, як існує багато способів визначення і написання програми для розв’язання конкретної задачі. Часто буває, що програмісти припускаются тих самих помилок і зустрічаються з тими самими проблемами.

Для будь-якої фірми, що розробляє програмні продукти, корисно збирати статистичні дані про помилки і розбивати їх на категорії. У результаті можна скласти

перелік найчастіше повторюваних помилок, щоб кожен програміст міг учитися на чужих помилках, перш ніж звертатися по допомогу.

1. *Помилки набору і переписування*. Значне число помилок — це прості помилки набору; наприклад, часто плутають цифру нуль і літеру О, змінюють місцями символи, з тієї чи іншої причини пропускають оператори і т. ін.

2. *Змінні величини без початкових значень*. Використання змінних величин до того, як проведено їх ініціалізацію.

3. *Використання змінних багатьма модулями*. Помилка відбувається тоді, коли два модулі намагаються використовувати ту саму змінну чи сферу пам'яті для збереження проміжних результатів. Майже неминуче в якійсь ситуації один з модулів зіпсує те, що інший записав у цю тимчасову пам'ять. Проблема значно ускладнюється в системах реального часу, де два модулі можуть переривати один одного, кликати один одного.

4. *Помилки керування і логіки*. Програміст намагається викликати модулі, звертання до яких не було передбачене, чи не викликає модулі, коли це потрібно було б зробити; або вибирає неправильний шлях під час умовного переходу і т. ін.

5. *Індексація з виходом за межі масиву*. Це помилка, що часто трапляється, програміст починає звертатися до полів, розташованих за межами масиву, що призводить до різних наслідків. Іноді програма переривається (якщо вона намагається звертатися до полів поза виділеною їй сферою пам'яті), чи вона потрапляє на дані іншого масиву і т. д. Це проблема, яку можна вирішити за допомогою додаткових засобів контролю помилок, що містяться в машинних мовах і трансляторах.

6. *Неправильне завершення програмних послідовностей*. Наприклад, процедура злиття декількох масивів. У програміста часто бувають ускладнення, коли читаються останні кілька записів одного файлу, у той час, як деякі з файлів уже вичерпані (з'явилися ознаки «кінець файла») і т. д.

7. *Непередбачені особливі ситуації*. Часто виявляється, що програміст складає програму тільки для очевидного випадку (наприклад, зазначеного в технічному завданні), не передбачаючи можливості виняткових чи помилкових ситуацій. У таких випадках результат поведінки необхідної реакції програми на помилки непередбачуваний.

8. *Неправильні зв'язки та інтерфейси*. Проблема, рішення якої є основною метою спадного тестування: від одного модуля до іншого передається неправильна інформація, модулі не засилають у пам'ять чи не відновлюють зміст своїх регістрів і т. д.

9. *Неправильні формати даних*. Програміст може припуститися помилок під час вибору масштабу (шкали) чи типу формату числових змінних своєї програми.

10. *Оперування константами, як параметрами*. У багатьох високорівневих мовах програмування оператори, що викликають підпрограми з аргументами у вигляді змінних і констант, можуть їх не розрізняти.

11. *Неправильне вживання складних булевих виразів*, де трапляються вкладені вирази, може призводити до помилок, якщо вони не інтерпретуються однозначно.

12. *Неправильне вживання вкладених умовних операторів*. Вкладені умовні оператори можуть викликати так багато помилок, що багато організацій забороняють

їхнє використання через неоднозначну інтерпретацію при непарній кількості IF та ELSE.

13. *Некоректний вихід з підпрограм*. Ця проблема властива головним чином мовам, де структура підпрограм недостатньо чітко формалізована.

14. *Звертання до областей даних після операторів Write*. Одна з типових проблем тут полягає в тому, що сфери даних чи буферні сфери оперативної пам'яті звільняються (чи іноді очищаються) операційною системою після копіювання за допомогою оператора Write. Будь-яка спроба програміста одержати значущі дані під час звертання до цих сфер може тому викликати велику плутанину.

15. *Проблеми, зв'язані з прапорцями*. Якщо програміст використовує велику кількість прапорців (ознак), то можуть існувати неприпустимі їх комбінації, так що неправильне встановлення будь-якого прапорця призведе до того, що програма протягом досить тривалого часу буде виконуватися до безглуздого «стану», перш ніж це виявиться.

16. *Непередбачені особливі випадки вводу/виводу*. Найбільш розповсюджена ситуація тут — це відсутність засобів, що передбачають відповідну обробку сигналів кінця файлу і помилки парності при операціях вводу/виводу.

17. *Відсутність аналізу кодів відповіді*. Операційна система реалізує більшість операцій вводу/виводу і ряд інших важливих службових функцій. Коли управління повертається прикладній програмі, формується деякий «код відповіді», який показує, наскільки успішно було виконано відповідну операцію. Багато програмістів забувають проаналізувати цей код чи припускають, що ніколи не буває помилок відповіді; тому якщо помилка відповіді усе-таки відбувається, їхня програма продовжує працювати так, начебто помилки немає.

18. *Лічильники занадто малої розрядності*. Типова проблема для багатьох економічних додатків. Якщо програміст виділив для лічильника обмежене місце, то легко може відбутися переповнення (випадок, аналіз якого найчастіше не передбачається) і скидання лічильників у нуль або втрата старших розрядів суми.

19. *Проблеми адресації*. У разі роботи мовою асемблера програміст може помилятися, використовуючи непряму адресацію, модифікацію і відносну адресацію.

20. *Помилки в складних виразах*. Ця проблема властива в основному науково-технічним додаткам. Навіть у разі використання дужок програмісти часто дістають у результаті обчислень зовсім не те, що передбачалося. Тому складні вирази краще розбивати на простіші і послідовно їх обчислювати.

21. *Проблеми вирівнювання*. Це типова проблема мови високого рівня. При пересиланні послідовностей символів, їхньому порівнянні чи виконанні визначених типів арифметичних операцій вирівнювання перемінних може відігравати велику роль, оскільки для різних типів даних вирівнювання виконується по лівому або правому краю.

22. *Вихід за верхню чи нижню межу програмного стека*. Перший випадок виникає, якщо в програміста вийшов більш глибокий рівень вкладеності чи рекурсії, ніж він планував спочатку (чи більш глибокий, чим допускає система); у другому випадку виявляється інша помилка — робиться спроба виходу з підпрограми, яка не була викликана.

23. *Помилкове використання загальних регістрів для базування*. З очевидних причин ця проблема, що доставляє найбільше всього турбот програмістам, що працюють мовою асемблера для машин з численними загальними регістрами і регістрами зсуву.

24. *Спроба працювати з даними як із програмою*. Це відбувається звичайно у випадку, коли програма робить перехід у сферу даних чи коли послідовність закодованих команд інтерпретується як дані. Це може бути також наслідком індексації з виходом за межі масиву, про що йшлося раніше. Якщо програма звернулася до даних після кінця масиву, це може бути область, зайнята правильними машинними кодами.

25. *Використання неправильної версії програми*. Це відбувається, якщо програміст працює з лістингом програми, об'єктним модулем і початковою програмою, що не відповідають один одному.

26. *Відсутність обмежника (роздільника)*. У багатьох мовах програмування як ознака кінця коментаря чи оператора програми використовується символ «крапка з комою» (;). В інших випадках обмежником оператора, групи операторів, чи макровизначення навіть цілої програми є спеціальне слово END (КІНЕЦЬ). Якщо програміст забуде вставити в потрібне місце крапку з комою чи оператор END, це може викликати помилку, наприклад, компілятор може сприйняти групу програмних операторів як частину довгого коментаря.

Методика локалізації помилок

Приведемо основні принципи і деякі прийоми локалізації помилок за тестовими даними.

Пропуск тесту на ПК (як і виконання звичайної програми) може закінчитися одним з наступних варіантів:

1) аварія в програмі;

2) зациклювання;

3) нормальне закінчення з видачею результатів (можливо, і правильних).

Якщо в тесті не передбачено друкування проміжних результатів, то для перших двох варіантів закінчення завжди, а для третього варіанта — в разі отримання неправильних результатів, постає завдання визначення місця або ділянки програми, де міститься помилка. Для перших двох варіантів здається, що місце помилки вже визначено: оператор, де сталася аварія, або оператор, що організує повторення циклу. Але насправді може виявитися, що аварія і зациклювання цей лише зовнішній вияв — симптом помилки, яка знаходиться зовсім в іншому місці програми. Локалізація в тому, власне, і полягає, що за симптомом помилки визначається дійсна причина і дійсне місце помилки.

У найпростішому випадку локалізованою ділянкою є певний фрагмент тексту програми, що виконується послідовно і що складається з кількох операторів або блоків. У реальній програмі, що містить цикли і умови, під ділянкою потрібно розуміти фрагмент процесу, що визначається програмою під час виконання.

Якщо немає помилок в еталонних результатах, в документації і описі чужих програм, то основною умовою швидкої локалізації помилок є чітке уявлення про

структуру програми, що налагоджується, взаємозв'язку блоків і змінних, особливостях реалізації. Велику допомогу при цьому можуть надати блок-схеми або узагальнені алгоритми, а також добре прокоментована і оформлена програма. Для локалізації помилок корисно вести щоденник налагодження, в якому фіксується результат кожної прогонки програми (блоку) на машині, зіставляється з еталонним результатом, підготовлюється наступний крок налагодження. Отримані лістинги програми розмічають, систематизують таким чином, щоб прив'язати їх до записів у щоденнику.

Щоденник також використовується для систематизації досвіду локалізації помилок, фіксації своїх типових помилок, а також для визначення трудомісткості налагоджувальних робіт, що допомагає правильно запланувати і організувати розробку програм у майбутньому.

Засоби локалізації помилок

За принципами роботи засоби локалізації були розділені на чотири типи: *аварійне друкування*, *друкування у вузлах*, *стеження*, *прокручування.*

Аварійне друкування — це друкування значень змінних у програмі в той момент її виконання, коли в ній виникає помилка, перешкоджаюча подальшому нормальному її виконанню, — аварія.

Конкретне місце включення в роботу аварійного друку визначається автоматично без використання інформації від програміста, який повинен тільки визначити список змінних, що видаються на друкування.

Завдяки аварійному друкуванню програміст отримує доступ до тих значень змінних, які вони мали в момент виникнення аварійної ситуації, їх аналіз дозволяє програмісту досить точно локалізувати помилку в програмі, а іноді й не одну.

Однак аварійне друкування не може бути використане як основний засіб локалізації помилок, оскільки відсутність аварії в програмі ще не говорить про те, що в ній немає помилок.

Аварійне друкування можна використати на стадії експлуатації програми як заміна друкування у вузлах. Після налагодження програми оператори друкування у вузлах з програми видаляються або вимикаються, а потім включається аварійне друкування, яке і залишається в програмі на випадок аварії в програмі на етапі супроводу.

Друкування у вузлах — це друкування проміжних значень змінних, що цікавлять програміста. Це дозволяє програмісту набувати значень змінних у довільні моменти роботи програми, а не тільки після виникнення аварійної ситуації в програмі. Необхідність друкування у вузлах виходить з того, що до аварії (до аварійного друкування) призводять лише досить грубі помилки в програмі, а для локалізації інших помилок потрібно видавати на друк значення змінних у тих місцях програми, які вважаються найближчими до присутніх у програмі помилок.

Змінні, що виводяться на друк, і вузли намічаються програмістом, в основному, на стадії алгоритмізації або після її закінчення, але до етапу програмування. Друкування у вузлах може бути здійснене звичайними засобами алгоритмічної мови.

Стеження є однією з форм контролю за ходом виконання налагоджувальної програми: це стеження за використанням у програмі вибраних змінних або за виконанням деяких операторів. Перше стеження назвемо арифметичним, друге логічним.

Стеження проводиться або по всій програмі, або на заданій програмістом ділянці. За арифметичного стеження звичайно здійснюється друкування значень вибраної програмістом змінної в ті моменти, коли їй привласнюється певне значення. У разі логічного стеження — друкування мітки вибраного оператора.

Прокручування призначається для отримання найбільш повної інформації про обчислення, що проводяться на якійсь ділянці програми (арифметична прокрутка) або про послідовність виконання всіх операторів на цій ділянці (логічне прокручування).

За арифметичного прокручування для програми, написаної алгоритмічною мовою, зазвичай друкуються результати виконання кожного оператора присвоєння на заданій ділянці програми.

За логічного прокручування зазвичай друкуються мітки або номери всіх операторів, що виконуються на заданій ділянці програми.

Стеження і прокручування (логічні й арифметичні) використовуються як доповнення до методу друкування у вузлах, в тих випадках, коли не вдається за друкованими результатами виявити помилку, або для відлагодження певної ділянці програми, в якій істотно переважають логічні оператори.

Контроль індексів. У всі транслятори включений відлагоджувальний режим, призначений для виявлення виходу значень індексів змінної за межі, вказані при описі масиву.

Друк зовнішньої пам'яті. Нарівні з друкуванням значень змінних, що знаходяться в оперативній пам'яті, часто доводиться роздруковувати і значення змінних, розташованих у зовнішній пам'яті. У цьому випадку використовуються службові програми (утиліти), призначені для роботи із зовнішніми носіями.

Виправлення помилок

Види виправлень. Виправляти програму доводиться багато разів: при виявленні помилок, для вставки відлагоджувального друку, внаслідок змін вимог замовника, у ході модернізації програми. Тому програміст повинен володіти засобами виправлень програми. Залежно від приналежності знайденої помилки до етапу алгоритмізації або програмування (кодування), її виправлення вимагає різного підходу.

Виправлення алгоритму. Якщо виявлена помилка була допущена на етапі алгоритмізації, то спочатку вносять зміни в алгоритм, а потім виправляють програму. Оскільки існує взаємозалежність різних частин програми, виправлення помилки в одному місці може призвести до виникнення помилок в інших місцях програми. Чим більш незалежні модулі, з яких складається програма, тим легше виправляти її алгоритм.

Після виправлення алгоритму виконується перевірка і прокручування внесених змін і всіх модулів програми, яких стосуються ці зміни. Після внесення виправлень в текст програми, для перевірки правильності внесених виправлень програму знов

тестують, причому для значних виправлень пропускають і ті тести, які вже проходили раніше.

Виправлення програми. Після виправлення алгоритму і при виявленні помилок етапу програмування проводиться перепрограмування зміненої ділянки алгоритму, це дає змогу зберегти відповідність текстів алгоритму і програми, що полегшує пошук і виправлення подальших помилок.

Існує два види виправлень: *порядкове і контекстне.*

Порядкове виправлення слугує для виправлення цілих рядків символьного тексту або деяких полів у них. Програміст зазначає номери рядків, що виправляються і, можливо, номери позицій в рядках.

Контекстне виправлення дозволяє замінити один і той же помилковий фрагмент тексту, що зустрічається в програмі кілька разів, на правильний текст.

Цей етап істотно спрощений за рахунок того, що транслятори мають вбудовані засоби для виявлення помилок етапу програмування з точною локалізацією помилки, її ідентифікацією, підказкою, як її виправити.

Виправлення можуть виконуватися як на стадії налагодження програми, так і на подальших етапах: при експлуатації або модернізації програми. При налагодженні програми виправлення торкаються, в основному, деталей алгоритму, а при експлуатації програми виправлення можуть торкатися і загальніших питань: формати зовнішніх даних; заміна констант на параметри, що змінюються, і т. д.

Великі виправлення. Якщо дотримуватися чіткої функціональності всіх модулів, то серйозна зміна якої-небудь функції програми під час модернізації зведеться до зміни або до заміни цілком одного з модулів програми. Якщо є значні доповнення до програми (або до якого-небудь модуля), то їх оформляють у вигляді окремих модулів.

Зміна форматів даних. Список і формати даних, що виводяться або вводяться, змінюються в ході налагодження і пробній експлуатації програми, якщо недостатньо продумане завдання на програмування або загальний проект, але звичайно після отримання перших результатів з’являються і нові побажання, і додаткові вимоги користувачів, що стосуються зміни інтерфейсу програми. Для того, щоб спростити внесення необхідних змін, треба виділити спеціалізовані модулі введення і виведення, і в них зосередити весь апарат спілкування із зовнішнім середовищем.

Зміна констант. Для того, щоб програма менше зазнавала виправлень при зміні її констант, потрібно заміняти змінними ті з констант, які можуть змінитися в ході експлуатації і модернізації. Перед використанням таких змінних у програмі ним необхідно присвоїти значення, рівне константі, що замінюється.

Приватним, але дуже важливим випадком параметрів-констант програми є розміри таблиць, що задаються граничними парами при описі масивів у програмі. Якщо не ввести змінні для позначення кордонів індексів, то надалі під час зміни розмірів таблиць, можливо, доведеться вносити виправлення в текст скрізь, де трапляється, наприклад, максимальне значення індексу: у заголовках циклу, в індексі змінних тощо.

У тому разі, коли змінні параметри програми використовуються в різних модулях, постає завдання такої побудови програми, щоб початкові значення параметрів можна було змінити лише в одному модулі.

Складання програми при тестуванні і налагодженні

При розробці великих, багатомодульных програмних комплексів використовують спеціальні способи їх складання при тестуванні.

Отримання надійних програмних комплексів неможливо забезпечити без попереднього тестування окремих модулів, оскільки тільки на рівні комплексу можливе управління комбінаторикою тестування: допускається одночасне тестування декількох модулів, полегшується задача налагодження (тобто пошуку і виправлення помилок) і т. д.

Збирання модулів у програмний комплекс може здійснюватися двома методами: монолітним, покроковим.

Монолітний метод збирання передбачає виконання автономного тестування кожного модуля, а потім їх одночасне збирання і тестування в комплексі.

Для автономного тестування будь-якого модуля потрібен модуль-драйвер (налагоджуючий модуль) і один або кілька модулів-заглушок (імітатори).

Драйвер — це модуль, що забезпечує виклик і передачу необхідних вхідних даних модулю, що тестується, і обробку результатів.

Заглушка — це модуль, що імітує функції модулів, що викликаються, що тестується.

Покрокове тестування передбачає послідовне підключення до набору вже протестованих модулів чергового модуля, що тестується.

Метод покрокового складання передбачає, що модулі тестуються не автономно, а підключаються почергово для виконання тесту до набору вже раніше протестованих модулів. При такому підході можливі два варіанти: спадне складання зверху-вниз або зростаюче знизу-вгору. При тестуванні зверху-вниз слід розробляти заглушки для всіх модулів, що викликаються. Процес продовжується доти, поки не буде складено весь комплекс.

При тестуванні знизу-вгору процес організується таким чином: тестуються модулі нижчого рівня. Для кожного з них потрібен драйвер. При зростаючому тестуванні треба буде розробляти драйвери, але не потрібні заглушки.

Порівнюючи монолітне і покрокове складання програм, можна відмітити ряд переваг і недоліків кожної з них.

Монолітне складання потребує великих витрат, оскільки передбачає додатково розробку драйверів і заглушок, тоді як у разі покрокового складання розробляються або тільки заглушки (зверху-вниз), або тільки драйвери (знизу-вгору).

При покроковому тестуванні раніше виявляються помилки в інтерфейсах між модулями, оскільки раніше починається збирання програми. При монолітному методі модулі «не бачать один одного» до останньої фази.

Налагодження програм при покроковому тестуванні здійснювати легше, оскільки більшість помилок інтерфейсу важко локалізувати в разі монолітного тестування. Однак безумовною перевагою монолітного методу складання є велика можливість розпаралелювання робіт.

Вибір способу складання програм залежить від конкретних умов розробки і особливостей програмного комплексу, що тестується.

Випробування та впровадження програм

Мета випробування програми — встановлення відповідності програми ЕОМ заданим вимогам і програмним документам. Розрізнюють *попередні і приймальні* випробування.

Мета попередніх випробувань — визначення реальних характеристик і показників якості дослідного зразка програмного виробу; перевірка їх відповідності вимогам, зазначеним у ТЗ, і допуск дослідного зразка до приймальних випробувань. Попередні випробування проводить підприємство-розробник за участю підприємства-замовника.

Мета приймальних випробувань — перевірка відповідності характеристик і показників якості дослідного зразка програмного виробу вимогам, вказаним в ТЗ, для вироблення рекомендацій з його вдосконалення, визначення найбільш ефективних режимів експлуатації, сфер застосування і придатності програмного виробу для промислового впровадження. Приймальні випробування проводить підприємство-замовник в умовах, зазначених у ТЗ, програмах і методиках випробувань.

На відміну від налагодження випробуванням піддають завершений продукт програмування, що пройшов усі стадії розробки й налагодження. При випробуваннях встановлюється факт відповідності програмного виробу своєму призначенню. Причиною незадовільних результатів випробування можуть бути помилки у постановці задачі (технічних умовах), специфікаціях, алгоритмах, програмах. Виявлення, локалізація і усунення помилок не є метою випробування. Якщо під час випробувань програмного виробу виявляється його невідповідність призначенню, то він повертається на доробку.

Упровадження програмного виробу — це сукупність заходів щодо введення в експлуатацію програмного виробу на ЕОМ користувача. Таким чином, упровадження — це один з видів послуг у системі обслуговування ЕОМ. Упровадження програмного виробу спеціалізованими підприємствами дозволяє прискорити і здешевити процес освоєння нових програмних виробів. Розрізнюють такі види впровадження програмного виробу: постачання; введення в експлуатацію за договорами з користувачем; настроювання програми; розробка і супроводження систем обробки даних на базі програмного виробу загального призначення і т. ін.

Контрольні питання

1. Зміст та засоби налагодження програм.

2. Переваги покрокового тестування згори-вниз.