**Практична робота № 11 Розроблення плану тестування програм та тестів**.

**Мета:** навчитися складати план тестування програм та тестів розробленої програми**.**

**Завдання:**

1. **ознайомитися з теоретичною частиною;**
2. **виконати тест 6 протягом заняття.**

**.**

**ТЕОРЕТИЧНА ЧАСТИНА**

Програм без помилок не існує. Практика доводить, що винуватцями помилок у програмах найчастіше бувають самі програмісти. Один із загальних законів практичного програмування полягає в тому, що жодна програма не дає бажаних результатів при першій спробі трансляції та виконання. Певне уявлення про справжні причини появи помилок у роботі програми дає таке процентне співвідношення джерел збоїв:

|  |  |
| --- | --- |
| Вхідні дані | 1% |
| Помилки користувача | 5% |
| Апаратура | 1% |
| Системне програмне забезпечення | 3% |
| Розробка системи | 15% |
| Програмування | 75% |

     Програміст повинен не тільки писати ефективні програми, але і знаходити в них усілякі помилки. Сучасна практика навчання програмуванню орієнтована, в основному, тільки на виконання програмістом першої половини своєї роботи. Це все одно, як навчати льотчика тільки зльоту, припускаючи, що з посадкою машини він якось розбереться сам, якщо буде виконувати всі операції зльоту в зворотному порядку.

     Існують два типи програмних помилок:

*синтаксичні помилки* - виникають через порушення правил мови програмування. Такі помилки зазвичай виявляються під час компіляції. Можуть бути виключені порівняно легко. Навіть якщо не переглядати текст програми можна бути впевненим, що компілятор на стадії трансляції знайде помилки і видасть відповідні попередження. Фактично пошук помилок здійснює компілятор, а їхнє виправлення - програміст;

*семантичні (логічні) помилки* - ті, що призводять до некоректних обчислень або помилок під час виконання (run-time error). Семантичні помилки усувають зазвичай за допомогою виконання програми з ретельно підібраними перевірочними даними, для яких відома правильна відповідь.

 Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування. При цьому потрібно завжди пам'ятати про фундаментальний висновок, зроблений професором Едсжером Дейкстрою у 1972 році: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!".

Ми коротко розглянемо принципи проведення ручного тестування, яке проводить програміст, на відміну від використання автоматизованих засобів тестування. **Суть ручного тестування** — полягає в тому, що програміст або тест-інженер приміряє на себе, неначе костюмчик, роль кінцевого користувача і, відповідно до тестового сценарію, перевіряє програмне забезпечення, проте не без участі логіки застосовує різноманітні техніки тест дизайну, планує, складає тест сценарії. **Головні мотиви такого тестування** — виявити поведінку програми, яка відрізняється від очікуваної, знизити кількість дефектів, забезпечити працездатність програми, оцінити зручність користування, і кінець-кінцем отримати **якісний** продукт, який буде задовольняти потребам користувачів. В програмі не повинно бути дій з невизначеним результатом.

**Підготовчий етап тестування**

На цьому етапі потрібно:

1. Здійснити аналіз вхідних даних і виявити значення, які можуть привести до неочікуваного або аварійного завершення. Наприклад, перевірка знаменника на нульове значення.
2. Визначити перелік функцій, які виконує програма, на зробити план перевірки кожної функцій, зокрема, що в результаті її виконання видається правильний результат. Так, для обчислювальних операцій потрібно провести "ручний" розрахунок даних, з результатами якого потрібно порівнювати отримані за виконання програми. Потрібно зауважити, що при відхиленні отриманого результату від очікуваного, повинно бути пояснення.
3. Перевірка функцій повинна здійснюватися за заздалегідь розробленим планом. Наприклад, програма виконує такий перелік функцій, який надається через меню:
4. Введення даних.
5. Виведення даних на консоль.
6. Виведення введених даних до файлу.
7. Читання файлу.
8. Модифікація даних.
9. Перезапис даних у файлі.
10. Дозапис даних до файлу.
11. Вихід з програми.

Якщо надається можливість обрати режим роботи програми введенням числа, яке відповідає потрібним діям (як у ваших останніх лабораторних роботах), то введення числа, яке не відповідає жодному з очікуваних режимів повинно викликати відповідні дії, наприклад, виведення відповідного повідомлення та повторний запит режиму (уникнення невизначеності).

Іншим прикладом тестової перевірки є визначення дій, які відбуваються, якщо запитано виведення даних на консоль, а дані не були ні введені в інтерактивному режимі, ні прочитані з файлу. Відповідно аналізується результат і, в разі потреби коригується програма.

Ще один приклад, перевірка дій програми в разі спроби прочитати файл що відсутній.

Всі ситуації, які можуть викликати неправильний або небажаний результат повинні бути визначені та оброблені програмістом. Останнє означає, що повинно видаватися відповідне повідомлення щодо виявленої нештатної ситуації (наприклад, введення даних не того типу або не з тими значеннями, що очікується, або відсутність файлу для оброблення).

1. Перевірка допустимості даних, наприклад, при введенні числових даних перевіряти, що дані дійсно числа, а якщо для значення є обмеження, то числа відповідають цим обмеженням; або, якщо очікується відповідь "так" чи "ні", введена інша відповідь.
2. Перевірка на можливість появи нескінченного циклу внаслідок некоректних даних.

Це тільки мінімальний перелік перевірок. Чим складніша програма, тим більше тестових випадків потрібно розглядати.