Оглавление

Глава 1

Принципы тестирования и отладки

Тестирование — выполнение программы с целью обнаружения ошибок.
Отладка — определение места ошибки и внесение исправлений в программу.
Ошибки в программе есть всегда.
Тест — это совокупность исходных данных и ожидаемых результатов.
Необходимо фиксировать выполненные тесты и реально полученные результаты.
Необходимо проверять что программа делает то, что нужно и что программа не делает того, чего не нужно.
После исправления программы необходимо повторное тестирование.
Ошибка — место в программе, где программст написал некорректный код.
Сбой — неверная работа в программе.
Код, вызывающий сбой и код, содержащий ошибку как правило не один и тот же.

```
1  a = 0;
2  if (x > 3)
3    a = 10;
4  b = 100 / a;
```

Четвёртая строка программы — место проявления ошибки. Местом нахождения ошибки могут быть:

- первая строка, где переменная инициализируется
- вторая строка с неправильным условием
- третья строка, где отсутствует ветка «иначе»
- четвёртая строка, где записано неправильное выражение

Глава 2

Как тестировать свою программу

При тестировании нужно относиться к своей программе как к чужой.

2.1. Критерии тестирования

2.1.1. Критерий покрытия путей

Нужно покрыть все возможные пути в программе.

```
1  if (a == 0)
2    std::cout << "1";
3  if (b == 0)
4    std::cout << "2";
5  if (c == 0)
6    std::cout << "3";
```

Сколько вариантов вывода?

```
1  while (a != 0)
2  {
3      int b;
4      std::cin >> b;
5      a += b;
6  }
```

Разных путей бесконечно много из-за непредсказуемости пользовательского ввода.
Поэтому критерий покрытия путей невозможно применить на практике.

2.1.2. Критерий покрытия ветвей

Нужно стараться покрыть все ветви программы тестами.

```
1  if (a == 0)
2    std::cout << "1";
3  if (b == 0)
4    std::cout << "2";
5  if (c == 0)
6    std::cout << "3";
```

Тест $a = 0, b = 0, c = 0$ покрывает все ветви.

```
1  if (a == 0)
2    std::cout << "1";
3  else
4    std::cout << "-1";
5  if (b == 0)
6    std::cout << "2";
7  else
8    std::cout << "-2";
9  if (c == 0)
10   std::cout << "3";
11 else
12   std::cout << "-3";
```

Нужно минимум 2 теста. Например, $a = 0, b = 0, c = 0$ и $a = 1, b = 1, c = 1$.

```
1 if (a != 0 || c != 0)
2   std::cout << c / a;
3 else
4   std::cout << "0";
```

Здесь обе ветки можно покрыть с помощью тестов $a = 1, c = 1$ и $a = 0, c = 0$. Однако, возможность получить ошибку деления на 0 не проверена с помощью теста $a = 0, c = 1$.

2.1.3. Критерий покрытия условий

Чтобы каждая часть условия получила истинное или ложное значение хотя бы один раз.

Предыдущий пример иллюстрирует необходимость покрытия разных комбинаций простых условий в составном.

Модифицированный критерий покрытия условий — критерий комбинаторного покрытия условий. Требует, чтобы хотя бы один раз выполнялась каждая комбинация простых условий.

2.1.4. Как придумывать тесты

Граничные значения: минимальные и максимальные значения параметров, максимальное время выполнения, максимальная используемая память.

Получить разные варианты вывода (например, «да» и «нет»)

Особенные для алгоритма значения: чтобы выполнялся один из критериев покрытия тестами.

2.2. Задания

Дана программа. Придумать тесты, записать в таблицу входные и выходные данные. Объяснить результаты.

Глава 3

Методы отладки

С помощью тестов можно обнаружить только место проявления ошибки.

Последовательность отладки: Найти тест, вызывающий сбой Уменьшить тест, чтобы происходил тот же самый сбой Локализовать место, содержащее ошибку

TODO: программа для демонстрации уменьшения теста

3.1. Отладочный вывод

TODO: программа для демонстрации

3.2. Утверждения

TODO: программа для демонстрации

3.3. Принципы отладки

- Обдумывать результаты каждого теста
- Не вносить случайные исправления
- Проверять тестами конкретные гипотезы
- Исправлять ошибки по очереди
- Постараться объяснить все видимые симптомы сбоя
- Исправление может внести новые ошибки

3.4. Отладчики

Глава 4

Интерактивный отладчик Visual Studio

- Пошаговое исполнение
- Исполнение «до курсора»
- Просмотр значений переменных
- Точки прерывания
- Точки трассировки
- Точки наблюдения
- Наблюдение за стеком

Глава 5

Наиболее часто встречающиеся ошибки

5.1. Использование неинициализированной переменной

Иногда может быть обнаружена на этапе компиляции. Часто приводит к неожиданному поведению программы, которое меняется от запуска к запуску.

5.2. Неверно инициализированные переменные

Если переменная объявлена глобально, она заполняется нулями, что не всегда подходит.

5.3. Выход за границы массива

Обращение к массиву очень сложно проконтролировать в C/C++.

5.4. Использование указателей на освобождённую память

Динамическая память может быть уже заполнена другими данными.

5.5. Ошибки с приоритетами операций

5.6. Неправильная вложенность условных операторов

```
1 if (A)
2   if (B)
3     X;
4 else
5   Y;
```

Глава 6

Конкурс по отладке программ