Программная инженерия

Барышникова Марина Юрьевна
МГТУ им. Н.Э. Баумана
Каф. ИУ-7
baryshnikovam@mail.ru

Лекция 7

Обнаружение ошибок в программе. Защитное программирование. Утверждения. Условная компиляция

Причины возникновения ошибок в ПО

Количество ошибок в программе зависит:

- от сложности той проблемы предметной области, для решения которой разрабатывается программа;
- от уровня квалификации команды разработчиков;
- от специфики языка программирования и инструментов, которые использовались при проектировании, написании и тестировании программы;
- от уровня зрелости тех сервисных функций и технологий, которые задействуются в ходе эксплуатации программы;
- от ряда других факторов

Виды ошибок в программах

- Ошибки в описании задачи (программа правильно решает неверную задачу)
- Ошибки в выборе алгоритма (использование неподходящего или неэффективного алгоритма)
- Ошибки анализа (связаны с неполным учетом возможных ситуаций, либо с неправильным решением задачи):
  - отсутствие задания начальных значений переменных;
  - неверные условия окончания цикла;
  - неверная индексация цикла;
  - неправильное указание ветви алгоритма для продолжения процесса решения задачи;
  - и др.

Виды ошибок в программах

- Ошибки общего характера (не зависящие от языка программирования):
  - ошибки из-за недостаточного знания или понимания программистом языка программирования или самой машины;
  - ошибки, допущенные при программировании алгоритма, когда команды, используемые в программе, не обеспечивают последовательности событий, установленной алгоритмом

Виды ошибок в программах

- Ошибки физического характера (вызываемые неверными действиями программиста):
  - пропуск некоторых операторов;
  - отсутствие необходимых данных;
  - непредусмотренные данные;
  - неверный формат данных

Обобщенная классификация ошибок

Вид ошибок	Пример
1. Неправильная постановка задачи	Правильное решение неверно сформулированной задачи
2. Неверный алгоритм	Выбор алгоритма, приводящего к неточному или неэффективному решению задачи
3. Ошибки анализа	Неправильное программирование алгоритма
4. Семантические ошибки	Непонимание порядка выполнения команды
5. Синтаксические ошибки	Нарушение правил, определяемых языком программирования
6. Ошибки при выполнении операций	Отсутствие, указаний на ограничивающие условия вычислений (деление на нуль и т.д.)
7. Ошибки в данных	Неудачное определение возможного диапазона изменения данных
8. Ошибки в документации	Документация пользователя не соответствует действующему варианту программы

Косвенные признаки наличия ошибок в программе

- Отсутствует уверенность в том, что программа начала выполняться
- Программа начала выполняться, но произошел преждевременный останов с выдачей или без выдачи сообщения о системной ошибке
- Программа начала выполняться, но зациклилась, о чем можно судить по ее чрезмерно долгой работе
- Программа выдала неправильную информацию

Большинство ошибок воспроизводимо, и это по крайней мере подтверждает наличие их в программе

Неповторяющиеся ошибки могут быть вызваны неправильными действиями пользователя, сбоем в работе оборудования, колебаниями питающего напряжения или тупиковыми ситуациями в операционной системе

Процесс обнаружения ошибки

- Точка обнаружения ошибки — это место в программе, где ошибка себя проявляет или становится очевидной. Точка обнаружения должна выявляться первой
- Точка происхождения ошибки — это место в программе, где возникают условия для появления ошибки

C=B/A

Отладочная информация

Это информация, выдаваемая на печать в процессе отладки программы, которая состоит из операторов, характеризующих результаты выполнения отладочных действий:

- эхо-печать всех введенных данных;
- печать информации о ходе вычислительных операций;
- печать информации о работе логической части программы

Выборочная печать

IF (X<0) WRITE ...

IF (I < 10 AND I > 15) WRITE …

IF (I/5*5 - I = 0) WRITE ...

IF (N = 1000) WRITE ...

IF DEBUGGING THEN ...,

где DEBUGGING – логическая переменная, которой присваивается значение true или false

Прослеживание логических ветвей

- ВХОД В ПОДПРОГРАММУ MAX_SUM
- ВЫХОД ИЗ ПОДПРОГРАММЫ MAX_SUM
- ВХОД В ПОДПРОГРАММУ MIN_KOL
- ПЕРЕХОД НА ВЕТВЬ "МЕНЬШЕ НУЛЯ"
- ОДНА ТЫСЯЧА ИНТЕРАЦИЙ
- НОРМАЛЬНЫЙ КОНЕЦ РАБОТЫ

Стратегия ввода в программу специальных отладочных операторов, предназначенных для выдачи данных на печать, обеспечивает получение четкого представления о том, что происходит при выполнении как арифметических, так и логических операций

Как правило, чем ближе к началу программы расположены такие вспомогательные операторы, тем меньше требуется в дальнейшем отладочных прогонов

Защитное программирование

Это метод организации программного кода таким образом, чтобы при работе системы последствия проявления ошибки в программе не приводили бы к сбоям

Защитное программирование предполагает такой стиль написания программ, при котором появляющиеся ошибки легко обнаруживаются и идентифицируются программистом

Фактически защитное программирование заключается во встраивании отладочных средств в программу. Средства отладки, предусматриваемые в исходной программе, называют стопорами ошибок

Принципы защитного программирования

- Общее недоверие к данным - входные данные каждого модуля должны тщательно анализироваться в предположении, что они ошибочны
- Немедленное обнаружение - каждая программная ошибка должна быть выявлена как можно раньше, что упрощает установление ее причины
- Изолирование ошибок - ошибки в одном модуле должны быть изолированы так, чтобы не допустить их влияния на другие модули

Главная идея защитного программирования в том, что если методу передаются некорректные данные, то его работа не нарушится, даже если эти данные испорчены по вине другой программы

Стив Макконелл

Рекомендации по реализации защитного программирования

- Проверяйте тип данных. Контролируйте буквенные поля (поля имен), чтобы убедиться, что они не содержат цифровых данных. Проверяйте цифровые поля на отсутствие в них буквенных данных
- Делайте проверку области значений переменных, чтобы удостовериться, например, что положительные числа всегда положительны
- Выполняйте контроль правдоподобности значений переменных, которые не должны превышать некоторых констант или значений других переменных. Например, начисляемые налоги и удержания не должны быть больше суммы, для которой они определяются
- Контролируйте итоги вычислений путем введения всюду, где это возможно, перекрестных итогов, контрольных сумм и счетчиков числа обрабатываемых элементов информации

Рекомендации по реализации защитного программирования

- Используйте автоматические проверки, такие, как контроль за переполнением, потерей значимости и состояниями файлов
- Проверяйте длину элементов информации, если она задана, например код почтового индекса
- Контролируйте закрепленные признаки, под которыми понимаются обязательные элементы полей или записей данных. Например, если определенный тип записей должен всегда содержать в определенных полях только числовые данные – это надо проверять
- Выполняйте проверку контрольных разрядов. Так как некоторые элементы данных могут иметь дополнительные контрольные цифры, существует возможность проверки правильности этой информации

Методы защитного программирования при обработке данных

Программа, построенная с применением приемов защитного программирования, должна:

- сообщать пользователю об области допустимых значений исходных данных при формулировке задачи или при вводе данных
- контролировать значения исходных данных при их вводе, сообщать о невозможности выполнения вычислений для недопустимых значений
- обеспечивать для каждой из подобластей допустимых значений соответствующие ей вычисления, которые могут отличаться:
  - типами данных, участвующих в вычислениях
  - алгоритмами и схемами вычислений
- контролировать промежуточные результаты вычислений, прекращать вычисления или изменять их порядок при обнаружении недопустимых ситуаций

Требования к обработке входных данных

- Проверяйте все данные из внешних источников
- Проверяйте значения всех входных параметров процедур и функций
- Решите, как обрабатывать неправильные входные данные

Альтернативные стратегии

- вернуть некое нейтральное значение
- заменить следующим корректным блоком данных
- вернуть тот же результат, что и в предыдущий раз
- подставить ближайшее допустимое значение
- записать предупреждающее сообщение в файл
- вернуть код ошибки
- вызвать процедуру/функцию — обработчик ошибки
- прекратить выполнение

Альтернативные стратегии

«мусор на входе – мусор на выходе»

«мусор на входе — ничего на выходе»

«мусор на входе — сообщение об ошибке на выходе»

Выбор подходящего метода обработки ошибки

- Корректность свойство программы удовлетворять поставленным требованиям, т.е. получать результаты, точно соответствующие решению задачи и требованиям к ее интерфейсу

Корректность предполагает, что нельзя возвращать неточный результат: лучше не вернуть ничего, чем неточное значение

- Устойчивость способность программы отслеживать ошибки при вводе и вычислении данных и сообщать об этих ситуациях, вместо выдачи неправильных результатов

Устойчивость требует всегда пытаться сделать что-то, что позволит программе продолжить работу, даже если это приведет к частично неверным результатам

Корректность или устойчивость?

- Обработка ошибок в общем случае может стремиться либо к большей корректности, либо к большей устойчивости кода;
- Выбор подходящего метода обработки ошибки зависит от приложения, в котором эта ошибка происходит;
- Для приложений, требовательных к безопасности, корректность предпочтительней устойчивости: лучше не вернуть никакого результата, чем неправильный результат;
- В потребительских приложениях устойчивость, напротив, предпочтительнее корректности: какой-то результат всегда лучше, чем прекращение работы

Выбор общего подхода к работе с некорректными данными — это вопрос архитектуры или высокоуровневого проектирования

Утверждения

Утверждение (assertion) — это код (обычно метод или макрос), используемый во время разработки, с помощью которого программа проверяет правильность своего выполнения

procedure Assert(Test: Boolean); *(Pascal)*
void assert(expression); *(C)*

Assert предназначен для документирования и проверки истинности допущений, сделанных при написании кода

Утверждения не предназначены для показа сообщений в промышленной версии. Они в основном применяются при разработке: информация, заключенная в Assert-е, - это способ уведомить разработчика о каких-то проблемах в программном коде или в интерфейсе программы. Так как утверждения удаляют при компиляции промышленной версии, в них нельзя помещать выполняемый код

Использование утверждений в программе для начисления зарплаты

```
#include <assert.h>

double Salory;  {Зарплата}
double Tax; {Налог}
double Payment; {Вознаграждение}

    printf ("Введите зарплату:  ");
    scanf ("%lf", &Salory);

    assert(Salory > 0) and (Salory < 100000);

    Tax=Salory*0.13;
    assert(Tax<Salory);
    printf ("Налог = %lf \n" , Tax);
    Payment = Salory-Tax;
    assert(Payment <Salory);
     printf ("Сумма на руки = %lf \n", Payment);
```

Когда отладка закончена, в начале программы достаточно добавить строку #define NDEBUG и все вызовы макроса assert будут игнорироваться

Случаи применения утверждений

Утверждения применяют для проверки того, что:

- значение входного (выходного) параметра попадает в ожидаемый интервал;
- файл (или поток) открыт (закрыт), когда процедура или функция начинает (заканчивает) выполняться;
- указатель файла (или потока) находится в начале (в конце), когда процедура или функция начинает (заканчивает) выполняться
- файл (или поток) открыт только для чтения, только для записи или для чтения и записи;
- значение входной переменной не изменяется в процедуре или функции;
- указатель ненулевой

Случаи применения утверждений

- массив (или другой контейнер), передаваемый в процедуру или функцию, может вместить по крайней мере N элементов;
- таблица инициализирована для помещения реальных значений;
- массив (контейнер) пуст (заполнен), когда процедура или функция начинает (заканчивает) выполняться;
- результаты работы сложного, хорошо оптимизированного метода совпадают с результатами метода более медленного, но написанного яснее

Общие принципы использования методов защитного программирования

- Используйте процедуры обработки ошибок для ожидаемых событий и утверждения для событий, которые происходить не должны

Обработчик ошибок обычно проверяет некорректные входные данные, утверждения — ошибки в программе

- Используйте утверждения для документирования и проверки предусловий и постусловий:
  - предусловия — это соглашения, которые должны быть выполнены до вызова подпрограммы, т.е. это обязательства вызывающего модуля перед модулем, который он вызывает;
  - постусловия — это соглашения, которые подпрограмма должна выполнить при завершении своей работы, т.е. это обязательства вызываемого модуля перед модулем, который их использует

Обработчик ошибок или утверждение?

Если для обработки аномальной ситуации служит обработчик ошибок, он позволит программе адекватно отреагировать на ошибку.

Если же в случае аномальной ситуации сработало утверждение, для исправления просто отреагировать на ошибку мало — необходимо изменить исходный код программы, перекомпилировать и выпустить новую версию ПО

Утверждения можно рассматривать как выполняемую документацию — с их помощью нельзя заставить работать программу, но можно документировать допущения в коде более активно, чем с использованием комментариев языка программирования

Условная компиляция

используется в том случае, когда удобно задать какую-то переменную и затем проверить, если эта переменная определена, то выполнить какую-то последовательность действий, а если нет, то не выполнять

Пример для C:

- #define DEBUG
- ...
- #if defined(DEBUG)
- // отладочный код
- ...
- #endif