Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА: **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ**

О Т Ч Е Т

по лабораторной работе № 3

Тема: Оценка эффективности и качества программы (Вариант 11)

Дисциплина: Технология разработки программных систем

Студент	ИУ6-42Б	03.06.24	А. П. Плютто
	(Группа)	(Подпись, дата)	(И. О. Фамилия)
Преподаватель		03.06.24	Е. К. Пугачёв
		(Подпись, дата)	(И. О. Фамилия)

Москва, 2024

Содержание

1. Задание

1.1. Цель работы

Цель данной работы — изучить известные критерии оценки и способы повышения эффективности и качества программных продуктов.

1.2. Задание по варианту

Написать программу вычисления суммы ряда $S = \frac{1}{4} - \frac{1}{16} + \frac{1}{96} - ... + (-1)^{n+1} \frac{1}{4^n \times n!}$ с точностью 0,0001. Ряд сходится и имеет множитель $m = -\frac{1}{4 \times n}$.

1.3. Код

```c
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <time.h>

int main()
{
    double s,a,m,eps,x;
    double *ms, *mss;
    int N = 1000, i,o;
    ms = (double*)malloc(N * sizeof(int));
    mss = (double*)malloc(N * sizeof(int));
    eps = 0.00001;
    ms[1] = 1.0/4;
    o = 1;
    s = 1.0/4;
    mss[1] = s;
    do {
        o++;
        m = -1/(4*o);
        ms[o] = ms[o-1]*m;
        s = s+ms[o];
        mss[o] = s;
    } while (abs(ms[o-1]-ms[o])>=eps);
    printf("s = %f\nn = %d",s,o);
    return 0;
}
```

Листинг 1 — *Заданный по варианту код*

2. Выполнение

Технологичность определяет качество проекта и, как правило, важна при разработке сложных программных систем. От технологичности зависят трудовые и материальные затраты на реализацию и последующую модификацию программного продукта. Факторами, которые определяют технологичность, являются: проработанность модели, уровень независимости модулей, стиль программирования и степень повторного использования кодов. Обычно в рамках лабораторных работ из-за ограниченного объема времени оценка этого качества не выполняется.

Эффективность программы можно оценить отдельно по каждому ресурсу вычислительной машины. Критериями оценки являются:

- Время ответа на вопрос или выполнения операции (оценивают для программ,работающихвинтерактивномрежимеиливрежимереального-времени);
- Объем оперативной памяти (важен для вычислительных систем, где оперативная память является дефицитным ресурсом);
- Объем внешней памяти (оценивают, если внешняя память является дефицитным ресурсом или при интенсивной работе с данными);
- Количество обслуживаемых терминалов и др.

Выбор критерия оценки эффективности программы существенно зависит от выполняемых ею функций. Например, если основной функцией является поиск данных, то уменьшить время выполнения можно, обеспечив избыточный объем оперативной памяти. В этом случае оценка памяти будет уже второстепенной задачей.

2.1. Добавление фиксаций времени

Добавим фиксации по времени: для этого используем функцию `clock()` из библиотеки `time.h`. Данная функция возвращает количество тиков процессора, которые прошли с момента запуска программы. Что бы получить из данного значения миллисекунды воспользуемся формулой: `(ticks * 1000) / CLOCKS_PER_SEC;`. В коде измерим 3 времени: время начала программы `t0` время начала цикла `t1` и время конца цикла `t2`, так мы можем получить время выполнения цикла и время выполнения всего кода до цикла. Считаем, что инициализация временных меток выполняется всегда за одно и то же время, таким образом убрав временные метки все оптимизации останутся эффективными. Код, получившийся после добавления временных меток:

В результате выполнения программы время ее работы составило 0.4 нс, выполнение цикла 0.1 нс. Данные цифры не очень информативны, поэтому используем линзу, чтобы получить более информативное время. Сделаем цикл, в котором прогоним код 1000000. Получаем 347.505 мс – общее время работы программы, таким образом точное время выполнения кода 0.347505 нс. Следует отметить, что в код я добавил очищение памяти, которое занимает некоторое время, но без него будет выделено большое количество памяти без очищения.

2.2. Улучшения качества кода

Качество программных продуктов может оцениваться следующими критериями:

- правильность — функционирование в соответствии с заданием;
- универсальность — обеспечение правильной работы при любых допустимых данных и содержание средств защиты от неправильных данных;
- надежность (помехозащищенность) — обеспечение полной повторяемости результатов, т. е. обеспечение их правильности при наличии различного рода сбоев;
- проверяемость — возможность проверки получаемых результатов;
- точность результатов — обеспечение погрешности результатов не выше заданной;
- защищенность — сохранение конфиденциальности информации;
- эффективность — использование минимально возможного объема ресурсов технических средств;
- адаптируемость — возможность быстрой модификации с целью приспособления к изменяющимся условиям функционирования;
- повторная входимость — возможность повторного выполнения без перезагрузки с диска (для резидентных программ);
- реентерабельность — возможность «параллельного» использования несколькими процессами.

Для улучшения качества кода введем несколько улучшений и исправлений:

1. Названия переменных: изменим названия переменных на более понятные, для улучшения читаемости кода.
    - `s` переименуем на `sum`
    - `ms` переименуем на `terms`
    - `mss` переименуем на `cumulative_sums`
    - `o` переименуем на `index`

2. Выделение Памяти: Добавим проверку успешного выделения памяти и очистку памяти.
3. Математическая библиотека: Включён заголовок `<math.h>` для использования `fabs` для вычисления абсолютного значения. `fabs` использован для уменьшения преобразования типов.
4. Читаемость кода: Улучшена читаемость кода за счёт добавления правильных отступов и пробелов.

После улучшения качества кода получаем следующий код:

2.3. Способы уменьшения времени выполнения

Время выполнения программы в первую очередь зависит от используемых в ней методов реализации операций. Важную роль играют циклические фрагменты с большим количеством повторений, поэтому по возможности необходимо минимизировать тело цикла.

При написании циклов рекомендуется:

1. Выносить из тела цикла выражения, не зависящие от параметров цикла;
2. Заменять «длинные» операции умножения и деления вычитанием и сдвигами;
3. Уменьшать количество преобразования типов в выражениях;
4. Исключать лишние проверки;
5. Исключать многократные обращения к элементам данных сложных структур (например, многомерных массивов, так как при вычислении адреса элемента используются операции умножения на значение индексов);
6. Исключать лишние вызовы предопределенных процессов (например, процедур, функций, методов классов и др.).

Посмотрим, насколько все рекомендации применимы к коду:

1. Все выражения, используемые в цикле зависят (иногда косвенно) от параметров цикла
2. Операцию умножения `o` на 4 можно заменить сдвигом вправо на 2, операцию деления заменить сдвигом нельзя, так как деление не целочисленное
3. Можно уменьшить количество преобразований типов в выражениях:

- `abs` заменяем на `fabs`, таким образом числа с плавающей точкой не будут переводиться в целочисленный тип, а будут сразу же сравниваться

4. Исключать лишние проверки мы не можем, так как проверка всего одна.
5. Все массивы одномерные, таким образом при формировании адреса не будут использоваться операции умножения.

При этом есть ошибка в самой реализации(не правильные индексы) и лишние массивы: нам нет необходимости находить все элементы ряда, нужна только сумма, поэтому цикл весь перепишем.

Суммируя все вышесказанное в пунктах 2.1-2.3 можно получить следующий код:

```
#include <stdio.h>
#include <time.h>
#include <math.h>

double get_current_time_ms() {
    clock_t ticks = clock();
    return (ticks * 1000.0) / CLOCKS_PER_SEC;
}

int main() {
    double start_time = get_current_time_ms();

    for (int i = 0; i < 1000000; i++){
        double epsilon, last, next, sum;
        epsilon = 0.00001;
        last = 1 << 2;
        next = last;
        do {
            last = next;
            next = next/4;
            sum += next;
        } while (fabs(last - next) >= epsilon);
    }
    double computation_end_time = get_current_time_ms();
             printf("start_time   =   %f\ncomputation_start_time   =
%f\ncomputation_end_time = %f\nComputation Time = %f\n", start_time,
computation_end_time, computation_end_time - start_time);
    return 0;
}
```

Листинг 2 — *Исправленный код*

Опять используем линзу для замера времени. В этот раз получили 83.374 мс. Тогда сама программа работает за 0.083374 нс.

2.4. Оценка эффективности

Критерий оценки	Исходная программа		Исправленная программа	
	Недостатки	Количественная оценка	Недостатки	Количественная оценка
Время выполнения	Лишние преобразования типов, деление на 4, из-за использования лишних массивов лишние присваивания	0.347505 нс	Удалены лишние массивы, заменена функция abs на fabs	0.083374 нс
Оперативная память	Оба массива лишние, не предусмотрены проверки на верное выделение памяти, не предусмотрена проверка на освобождение памяти	8032 байт	Убраны массивы и не нужные переменные для их индексации	16 байт

2.5. Оценка качества

Результаты оценки	Критерий оценки			
	Правильность	Универсальность	Проверяемость	Точность результатов
Недостатки	Ошибка при использовании индексов	Входными данными является точность, выделяется фиксированное количество памяти, значит при большой точности массивы могут быть переполнены	Выводится результат и индекс, до которого считается сумма. Для лучшей проверяемости необходимо знать еще последние 2 элемента массива и их разность	Результат неверный, так как в начале ошибка в индексах массива
Оценка	2	2	3	0

3. Заключение

В результате проведенных экспериментов были выполнены замеры времени работы программы, оценки памяти, а также предложены способы повышения эффективности и качества программы.