|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**ОТЧЕТ**

|  |  |
| --- | --- |
| **По лабораторной работе №3** |  |

**Название:** Оценка эффективности и качества программ.

**Дисциплина:** Технологии разработки программных систем.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ-42б |  |  | С.В. Астахов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  |  |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2021

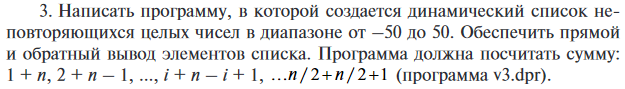
3 вариант

Введение

В настоящее время перед разработчиками программного обеспечения стоит задача создания эффективных, технологичных и качественных программ. Данная задача усложняется тем, что четких и универсальных рекомендаций для оценки указанных свойств не существует. Однако, на данный момент накоплен некоторый опыт, который может быть использован разработчиками.

**Цель работы:** изучить основные критерии оценки и способы повышения эффективности и качества программных продуктов.

Задача



Исходный код программы:

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <time.h>

int main()

{

int N = 10, i,j,k=0,f = 0;

double \*a;

double pr = 1;

a = (double\*)malloc(N \* sizeof(int));

srand(time(NULL));

a[0] = - 100 + rand()%(100 + 100 + 1);

for (i=1; i<N; i++){

while (f == 0){

a[i] = - 100 + rand()%(100 + 100 + 1);

for (j =0; j<i;j++){

if (a[i] != a[j]) k++;

}

if (k = = i) f = 1;

k=0;

}

f = 0;

}

printf("Исходный массив:\n");

for (i=0; i<N; i++){

printf("%f\n ", a[i]);

}

for (i=0; i<N; i++){

pr \*= a[i];

}

printf("Произведение:%f\n",pr);

return 0;

}

Исправления программы

1) Исходная программа работает некорректно, выдавая ошибки, так как при выделении памяти под массив адресуется память под N элементов типа int (4 байта на элемент), а записываются в массив элементы типа double (8 байт на элемент), следовательно, в ходе цикла программа выходит за пределы выделенной памяти.

Исправим эту ошибку, выделив память под N элементов типа double.

a = (double\*)malloc(N \* sizeof(double));

Программа стала работать корректно. Вывод программы:

Default array:

-98.000000

-22.000000

32.000000

-97.000000

-79.000000

89.000000

26.000000

-6.000000

-10.000000

-50.000000

Product:-3670136101632000.000000

Не меняя кода, реализующего основную логику программы, напишем «обертку», позволяющую считать эффективность программы по времени и использованию памяти при выполнении ее 100 000 раз:

#include <chrono>

#include <windows.h>

#include <Psapi.h>

using namespace std;

using namespace std::chrono;

int main()

{

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

PROCESS\_MEMORY\_COUNTERS memCounter1;

BOOL result = K32GetProcessMemoryInfo(GetCurrentProcess(), &memCounter1, sizeof(memCounter1));

**for (int cnt = 0; cnt < 100000; cnt++) {**

**// Исходная программа**

**}**

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

duration<double> time\_span = t2 - t1;

**std::cout << "It took me " << time\_span.count() << " seconds.";**

std::cout << std::endl;

PROCESS\_MEMORY\_COUNTERS memCounter2;

result = K32GetProcessMemoryInfo(GetCurrentProcess(), &memCounter2, sizeof(memCounter2));

**std::cout << "Memory leak: " << memCounter2.WorkingSetSize - memCounter1.WorkingSetSize << std::endl;**

std::cout << "\n(press any key to exit)\n";

std::getchar();

return 0;

}

Проведем первичное измерение характеристик программы на 100 000 итераций:

Затраченное время: 0.177567 секунд

Затраты памяти: 13 303 808 байт

2) Исходная задача генерирует значения в диапазоне [-100;100]. Исправим их на значения [-50;50]

3) Исходная программа генерирует динамический массив вместо списка. Используем список.

4) Исходная программа считает произведение элементов вместо заданной суммы.

5) Программа работает с дробными числами, задание требует использования целых чисел. Исправим это.

Исправленная версия программы:

int N = 10, i, j, k = 0, f = 0;

double \*a;

double pr = 0;

Node\* list = new Node;

Node\* p;

Node\* pb;

p = list;

srand(time(NULL));

p->val = -50 + rand() % (50 + 50 + 1);

p->next = nullptr;

for (i = 1; i < N; i++) {

while (f == 0) {

//std::cout << "while tick \n";

p->next = new Node;

p = p->next;

p->val = -50 + rand() % (100 + 100 + 1);

p->next = nullptr;

pb = list;

while (pb != nullptr) {

//std::cout << "while not null tick \n";

if (pb->val != p->val) k++;

pb = pb->next;

}

if (k == i) f = 1;

k = 0;

}

f = 0;

}

printf("List:\n");

pb = list;

while (pb != nullptr) {

std::cout << pb->val << "\n";

pb = pb->next;

}

i = 0;

pb = list;

while (pb != nullptr) {

i++;

pr += i + (pb->val) - i + 1;

pb = pb->next;

}

std::cout << "Sum: " << pr << "\n";

Вывод программы:

List:

-29

-7

45

-31

-12

0

24

-41

-25

-24

Sum: -90

Проведем первичное измерение характеристик программы на 100 000 итераций:

Затраченное время: 1.135 секунд

Затраты памяти: 56 516 608 байт

Повышение эффективности программы

В ходе программы выделяется динамическая память под элементы списка, однако она не очищается. Исправим это, добавив освобождение памяти в конец программы.

Проведем измерение характеристик программы (для 10 000 записей):

Затраченное время: 0.135256 секунд

Затраты памяти: 823 296 байт

Из расчета на 100 000 записей

Затраченное время: 1.35256 секунд

Затраты памяти: 8 232 960 байт

Как видно, временные характеристики программы изменились **незначительно**, затраты памяти же уменьшились **в несколько раз**.

Фрагмент усовершенствованной программы:

for (i = 1; i < N; i++) {

while (f == 0) {

//std::cout << "while tick \n";

p->next = new Node;

p = p->next;

p->val = -100 + rand() % (101);

p->next = nullptr;

pb = list;

f=1;

while (pb != nullptr) {

//std::cout << "while not null tick \n";

if (pb->val != p->val) k++;

pb = pb->next;

}

if (k == i) f = 1;

k = 0;

pb = pb->next;

}

}

f = 0;

}

printf("List:\n");

pb = list;

while (pb != nullptr) {

std::cout << pb->val << "\n";

pb = pb->next;

}

std::cout << "\nSum: " << pr << "\n";

p = list;

pb = list->next;

**while (pb != nullptr) {**

**free(p);**

**p = pb;**

**pb = pb->next;**

**}**

Повышение универсальности и проверяемости программы

1) Исходная программа не дает пользователю возможности задать длину списка. Исправим это (при этом будем проверять, что введенные данные — целое число в диапазоне [1; 100] ).

2) Программа не выдает пользователю элиенты, участвующие в подсчете суммы. Добавим дополнительные операции вывода, отображающие их.

Фрагмент усовершенствованной программы:

std::cout << "Enter list length (N <= 100)\n";

inputN = scanf\_s("%d", &N);

if (N > 0 && N <= 100 && inputN == 1) {

**// ввод списка**

}

else {

std::cout << "\nN is invalid\n";

}

printf("List:\n");

pb = list;

while (pb != nullptr) {

std::cout << pb->val << "\n";

pb = pb->next;

}

std::cout << "\n\nSum elems:";

pb = list;

while (pb != nullptr) {

pr += i + pb->val - i + 1;

**printf("\nf(n) = i+n-i+1 = %d + (%d) - %d + 1 = %d", i, pb->val,i, i + pb->val - i + 1);**

pb = pb->next;

}

std::cout << "\nSum: " << pr << "\n";

Пример работы усовершенствованной программы:

Enter list length (N <= 100)

20

List:

-12

-36

5

-2

14

-36

8

-43

10

-48

31

20

20

-27

18

46

4

12

-23

49

Sum elems:

f(n) = i+n-i+1 = 20 + (-12) - 20 + 1 = -11

f(n) = i+n-i+1 = 20 + (-36) - 20 + 1 = -35

f(n) = i+n-i+1 = 20 + (5) - 20 + 1 = 6

f(n) = i+n-i+1 = 20 + (-2) - 20 + 1 = -1

f(n) = i+n-i+1 = 20 + (14) - 20 + 1 = 15

f(n) = i+n-i+1 = 20 + (-36) - 20 + 1 = -35

f(n) = i+n-i+1 = 20 + (8) - 20 + 1 = 9

f(n) = i+n-i+1 = 20 + (-43) - 20 + 1 = -42

f(n) = i+n-i+1 = 20 + (10) - 20 + 1 = 11

f(n) = i+n-i+1 = 20 + (-48) - 20 + 1 = -47

f(n) = i+n-i+1 = 20 + (31) - 20 + 1 = 32

f(n) = i+n-i+1 = 20 + (20) - 20 + 1 = 21

f(n) = i+n-i+1 = 20 + (20) - 20 + 1 = 21

f(n) = i+n-i+1 = 20 + (-27) - 20 + 1 = -26

f(n) = i+n-i+1 = 20 + (18) - 20 + 1 = 19

f(n) = i+n-i+1 = 20 + (46) - 20 + 1 = 47

f(n) = i+n-i+1 = 20 + (4) - 20 + 1 = 5

f(n) = i+n-i+1 = 20 + (12) - 20 + 1 = 13

f(n) = i+n-i+1 = 20 + (-23) - 20 + 1 = -22

f(n) = i+n-i+1 = 20 + (49) - 20 + 1 = 50

Sum: 30

Пример работы усовершенствованной программы при некорректных входных данных:

Enter list length (N <= 100)

-3

N is invalid

Таблица 1 — оценка эффективности

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Критерий* | *Исходная программа* | | *Усовершенствованная программа* | | |
| *Комментарий* | *Количественная оценка* | | *Комментарий* | *Количественная оценка* |
| *Время* | Было сделано предположение о неоптимальности цикла проверки уникальности | 1.135 секунд на 100 000 итераций | | Не удалось повысить эффекстивность цикла проверки уникальности | 1.35256 секунд на 100 000 итераций |
| *Оперативная память (результаты измерений* ***для процесса****)* | Память, выделенная под список, **не** осовобождается | 56 516 608 байт на 100 000 итераций | | Память, выделенная под список, осовобождается | 8 232 960 байт  на 100 000 итераций |
| *Внешняя память* | Не используется | - | | Не используется | - |

Таблица 2 — оценка качества программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Оценка* | *Правильность* | *Универсальность* | *Проверяемость* | *Точность результатов* |
| *Исходная программа* | Программа работает с ошибками из-за некорректного выделения памяти, вместо списка используется массив, неверно задан интревал значений элементов списка, считается произведение вместо суммы, программа работает с действительными числами | Программа не позволяет задавать длину списка | Программа выводит исходные данные и результаты, но не выводит промежуточные, ее можно проверить, но это может быть затруднительно | Программа работает с целыми числами, особых требований точности нет |
| *оценка* | 2/5 | 3/5 | 4/5 | - |
|  | Ошибка выделения памяти исправлена, используется список, исправлен интервал значений элементов, посчитана исходная сумма, программа работает с целыми числами | Программа позволяет задавать длину списка. Позволяет выводить список в двух напралениях | Программа выводит как исходные данные и результаты, так и промежуточные подсчеты | Программа работает с целыми числами, особых требований точности нет |
| *оценка* | - | - | - | - |

Вывод: в ходе данной работы были изучены методы оценки и повышения эффективности и качества программ.