Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Специальность: Программная инженерия

Курс: 1

Группа: 10

Дисциплина: Основы алгоритмизации и программирования

 Реферат

На тему « Сравнение методов передачи данных в функции»

Выполнил:

Студент: Сегренёв Кирилл Сергеевич

Проверил:

Преподаватель: Белодед Н.И

2023, Минск

Содержание

[**ВВЕДЕНИЕ** 3](#_Toc153996381)

[**АЛГОРИТМ** 4](#_Toc153996382)

[**КОД ПРОГРАММЫ** 5](#_Toc153996383)

[**СРАВНЕНИЕ ВРЕМЕНИ ВЫПОЛНЕНИЯ** 7](#_Toc153996384)

[**ДИАГРАММА** 8](#_Toc153996385)

[**ПЛЮСЫ И МИНУСЫ** 9](#_Toc153996386)

[**ВЫВОД** 14](#_Toc153996387)

**ВВЕДЕНИЕ**

На лекции 19.12 преподаватель озвучил следующее задание: Сравнить скорость выполнения функций, в которые передаются данные:  
1. По значению

2. По указателю

3. По ссылке

Построить диаграммы и отметить плюсы и минусы каждого из способов.

**АЛГОРИТМ**

Для того, чтобы найти время выполнения своих функций с передаваемыми параметрами, я использовал класс high\_resolution\_clock из библиотеки <chrono>.

Я решил взять операцию возведения в степень с помощью функции pow() из библиотеки <cmath>, т.к она считается одной из самых медленных в выполнении.

Так же, чтобы было легче понять, какая функция выполнилась быстрее, а какая медленнее, я перевел обычные секунды в наносекунды, где уже отчетливо видна разница в скорости выполнения функций.

Для достижения более точных результатов выполним программу 3 раза и возьмем среднее арифметического каждого из времен, которые сравним между собой.

**КОД ПРОГРАММЫ**

#include <iostream>

#include <chrono>

#include <cmath>

using namespace std;

// Сравнение по значению

double value(double a, double b) {

return pow(a, b);

}

// Сравнение по указателю

double pointer(const double\* a, const double\* b) {

return pow(\*a, \*b);

}

// Сравнение по ссылке

double reference(const double& a, const double& b) {

return pow(a, b);

}

int main() {

setlocale(LC\_ALL, "ru");

double value1 = 19400.23;

double value2 = 102350.525;

auto start\_value = chrono::high\_resolution\_clock::now();// Фиксируем время начала выполнения операции

double result\_value = value(value1, value2); // Вызываем функцию для выполнения операции

auto end\_value = chrono::high\_resolution\_clock::now();// Фиксируем время окончания выполнения операции

// chrono::high\_resolution\_clock::now() нужно ставить два раза (до операции, которое мы хотим проверить на скорость выполнения, и после нее)

chrono::duration<double, nano> difference\_value = end\_value - start\_value; // Вычисляем разницу между данными начальным и конечным значением времени

// Это и получается наша скорость выполнение функции

cout << "Передача по значению: " << difference\_value.count() << " наносекунд." << endl; // Здесь мы выводим на экран значение нашей переменной используя метод .count()

// Аналогичные действия мы проводим с указателем и ссылкой

auto start\_pointer = chrono::high\_resolution\_clock::now();

double result\_pointer = pointer(&value1, &value2);

auto end\_pointer = chrono::high\_resolution\_clock::now();

chrono::duration<double, nano> difference\_pointer = end\_pointer - start\_pointer;

cout << "Передача по указателю: " << difference\_pointer.count() << " наносекунд." << endl;

auto start\_reference = chrono::high\_resolution\_clock::now();

double result\_reference = reference(value1, value2);

auto end\_reference = chrono::high\_resolution\_clock::now();

chrono::duration<double, nano> difference\_reference = end\_reference - start\_reference;

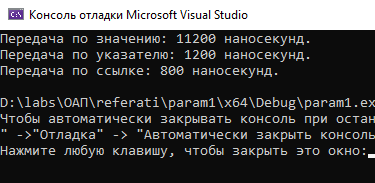
cout << "Передача по ссылке: " << difference\_reference.count() << " наносекунд." << endl;

return 0;

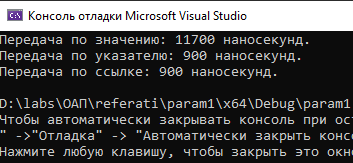
}

**Output:**

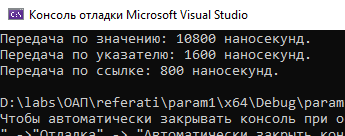
1.



2.



3.



**СРАВНЕНИЕ ВРЕМЕНИ ВЫПОЛНЕНИЯ**

Вычислим среднее значение времени выполнения функции, в которую передавали переменные:

1.По значению

(11200 + 11700 + 10800) / 3 = 11233 наносекунд

2. По указателю

(1200 + 900 + 1600) / 3 = 1233 наносекунд

3. По ссылке

(800 + 900 + 800) / 3 = 833 наносекунд

Из полученный значений мы видим, что самая быстрая по выполнению функция оказалась та, в которую передавали значения по ссылке. Это связано с тем, что ссылка не копирует данные переменных(как это делает передача по значению), и ссылка не требует разыменования(что, в свою очередь, требует указатель)

**ДИАГРАММА**

Из диаграммы видно, какая большая разница(почти в 10 раз) между передачей параметра по значению и по указателю/ссылке.

В наносекундах данная разница не сильно приведет к потере производительности, но если брать функции с большим телом, в котором много сложных операций, передачу по значению будут использовать меньшее число специалистов, т.к данный способ будет дольше обрабатывать информацию. Из этого следует, что предпочтение отведется передаче переменной по указателю либо по ссылке.

**ПЛЮСЫ И МИНУСЫ**

В данном разделе будут описаны плюсы и минусы использования каждого способа передачи переменных в функции.

**По значению**

Плюсы:

1. Передача аргументов по значению гарантирует, что любые изменения, производимые внутри функции, не затрагивают оригинальные переменные, переданные в функцию. В свою очередь это может быть и минусом.

Пример:

#include <iostream>

int sumParam(int u, int y) {

return ++u + ++y;

}

int main() {

int a = 5, b = 10;

int c = sumParam(a, b);

std::cout << c << ' ' << a << ' ' << b << std::endl;

return 0;

}

Данная программа выполняется увеличение на один каждой переменной, передаваемой в функцию, и слаживает их. Можно подумать, что в основной функции переменный a и b должны тоже увеличиться на 1, но нет. Выведем на экран сумму, которая вычислялась в функции sumParam(). Сумма равна 17, а переменные a и b в функции main() остались с такими же значениями, как и были инициализированы.

1. Простота

Передача по значению обычно проще в использовании, поскольку не требует явного управления памятью или указателями. Разработчику не нужно беспокоиться о том, какие изменения могут произойти внутри функции, и это делает код более понятным и поддерживаемым.

Минусы:

1. Время выполнения

Копирование значений переменных, особенно когда функцию вызывают много раз, может привести к сильному снижению производительности, а уже это в свою очередь приведет к более долгому выполнению участка кода.

1. Если использовать большие структуры и сложные операции, то передача по значению может быть неудобным решением из-за затрат на копирование данных.

**По указателю**

Плюсы:

1. Указатель указывает на конкретный адрес ячейки памяти, из-за этого значение переменной, которое содержится в этой ячейке, может изменяться.

Пример:

#include <iostream>

int sumPoint(int\* u, int\* y) {

return ++(\*u) + ++(\*y);

}

int main() {

int a = 5, b = 10;

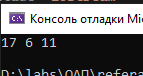
int c = sumPoint(&a, &b);

std::cout << c << ' ' << a << ' ' << b << std::endl;

return 0;

}

В данной программе у нас есть переменные a = 5 и b = 10, которые при заходе в функцию увеличиваются на 1 и складываются между собой. При возвращении в основную функцию мы можем видеть, что наши переменные изменились( a = 6, b = 11). Это произошло из-за того, что мы не копируем наши данные, а указываем на конкретный адрес памяти, где находится значение переменной.



Что происходит в это время в памяти:

b

10

234159

5

234149

a

main()

234159

y

234149

u

sumPoint()

Переменным u и y присваиваются адреса переменных в памяти, с помощью операции разыменования мы производим операции над данными по этому адресу.

1. Эффективность

При передаче указателя в функцию избегается ненужное копирование данных, которое может нагружать память и процессор. Из-за этого увеличивается скорость выполнения программы

1. Использование динамических массивов

Указатель позволяет использовать динамический массив в функции, что может позволить более гибко управлять памятью, ограничивать ее для большей оптимизации.

Минусы:

1. Утечки памяти и переполнение буфера

Неправильное обращение к указателем может привести к перезаписи важных данных, некорректное обращение к памяти, утечки памяти.

1. Сложность использования

Специалист должен полно владеть знаниями указателей, чтобы понимать и интегрировать их в свой код.  
  
3. Производительность  
  
Хоть скорость выполнения и меньше, чем при передаче данных по значению, но все же больше, если сравнивать с передачей данных по ссылке. Это связано с тем, что переменные, которые получают адрес, нужно разыменовывать, что приводит к более медленной работе, чем при использовании ссылок.

**По ссылке**

Плюсы:

1. Передача по ссылке является более производительной по сравнению с передачей по значению и указателю. Из-за того, что данные не копируются и не разыменовываются, данный способ является одним из самых производительных.
2. Передача по ссылке так же, как и указатель, изменяет переменные в главной функции, если над ними происходили операции. Это можно отнести и к плюсы и к минусу(смотря какая у нас ситуация)
3. Синтаксис проще, чем у указателей, но чуть труднее в понимании, чем у значений. Это связано с тем, что ссылка не работает непосредственно с памятью, а ссылается на элемент в ней.

Минусы:

1. В языке C, в отличии от C++, нет поддержки ссылочного типа.
2. Ссылку нельзя использовать для работы с динамической памятью, как это можно сделать с указателем. Это уменьшает гибкость в использовании ссылок, особенно когда работа с динамической памятью является ключевой частью программы
3. В некоторых случаях использование ссылок может сделать код менее ясным, особенно для тех, кто не знаком с их концепцией.
4. Ссылки не могут быть переопределены.

**ВЫВОД**

В проведенном анализе быстродействия программ было замечено, что способ передачи данных по ссылке выполнялся быстрее, что может быть связано с оптимизациями, предоставляемыми компилятором и особенностями передачи данных.

Ссылки избегают копирование данных и разыменовывания указателей.  
  
Важно помнить, что эффективность каждого метода может зависеть от конкретно поставленной задачи, от объема данных, от их типа и тд.   
  
В ситуациях, когда высокая читаемость кода или возможность модификации данных внутри функции являются важными факторами, рекомендуется рассмотреть иные методы передачи данных.

Таким образом, при разработке программы необходимо достигать баланса между производительностью и читаемостью кода, учитывая особенности конкретной задачи.