Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Перегруженные функции»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2023, Минск

**Содержание**

1. [Предисловие . . . 3](#Предисловие)
2. [Понятие перегруженной функции . . . 4](#Понятие_перегруженной_функции)
3. [Перегрузка функции с параметром-ссылкой . . . 5](#Перегрузка_функции_с_параметром_ссылкой)
4. [Перегрузка функции и параметры-константы . . . 6](#Перегрузка_функции_и_параметры_константы)
5. [Когда не нужно перегружать функции . . . 7](#Когда_не_нужно_перегружать_функции)
6. [Выводы . . . 8](#Выводы)
7. **Предисловие**

В данном реферате рассмотрено понятие перегруженной функции, также перегрузки функций с разными параметрами, когда не стоит перегружать функцию.

1. **Понятие перегруженной функции**

Под перегрузкой функции понимается, определение нескольких функций (две или больше) с одинаковым именем, но различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом. Компилятор распознает перегруженные функции только по получаемым параметрам. Если две функции имеют одинаковые имена, одинаковое количество и типы параметров, но возвращают разные значения, то такие функции считаются одинаковыми. В этом случае компилятор выдаст ошибку. Перегрузка функций нужна для того, чтобы избежать дублирования имён функций, выполняющих сходные действия, но с различной программной логикой.

Рассмотрим на примере функции, которая в зависимости от переданных параметров выводит разные сообщения:

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <stdio.h>

#include <conio.h>

using namespace std;

void vivod\_type(int a) {

cout << "Эта функция работает, если передано 1 целое число" << endl;

}

void vivod\_type(double a) {

cout << "Эта функция работает, если передано 1 вещественное число" << endl;

}

void vivod\_type(double a, double b) {

cout << "Эта функция работает, если передано 2 вещественных числа" << endl;

}

void vivod\_type(int a, int b) {

cout << "Эта функция работает, если передано 2 целых числа" << endl;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

vivod\_type(1);

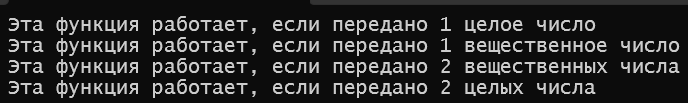
vivod\_type(1.5);

vivod\_type(1.6, 1.9);

vivod\_type(1, 10);

}

Результат выполнения:



Мы можем увидеть, что при разных передаваемых параметрах, мы получаем разный результат.

1. **Перегрузка функции с параметром-ссылкой**

При перегрузке функций с параметрами-ссылками следует учитывать, что параметры типов int и int& не различаются при перегрузке. Например, два следующих объявления:

int summa(int &x, int &y) {

return x + y;

}

int summa(int x, int y) {

return x + y;

}

Они не считаются разными версиями функции summa. В окне «Список ошибок» мы будем наблюдать ошибку:

#include <iostream>

#include <iomanip>

#include <Windows.h>

using namespace std;

int summa(int &x, int &y) {

return x + y;

}

int summa(int x, int y) {

return x + y;

}

int main() {

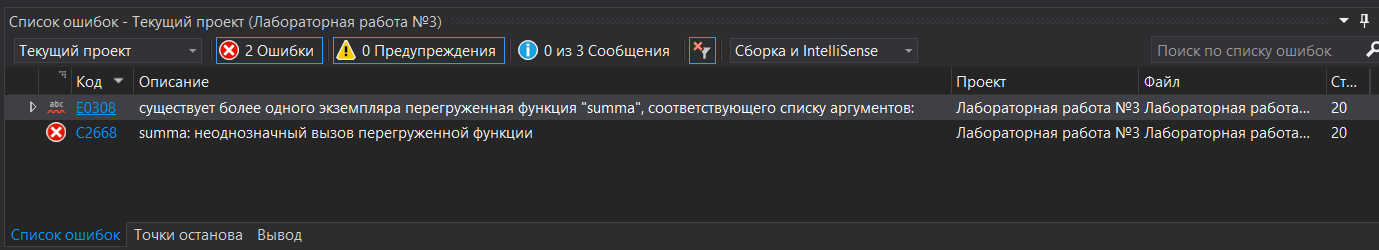
SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a = 1, b = 4;

cout << summa(a, b);

}



1. **Перегрузка функции и параметры-константы**

При перегрузке функций константный параметр отличается от не константного параметра только для ссылок и указателей. В остальных случаях константный параметр будет идентичен не константному параметру. Например, следующие два объявления при перегрузке различаться НЕ будут:

int chislo\_v\_kube(int a) {

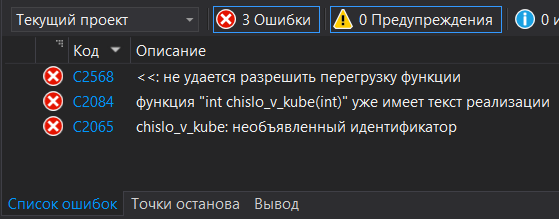
return a \* a \* a;

}

int chislo\_v\_kube(const int a) {

return a \* a \* a;

}



Во втором объявлении компилятор игнорирует оператор **const.** Но если мы будем принимать указатель на число, то уже будет разница.

Пример:

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include <stdio.h>

#include <conio.h>

using namespace std;

int chislo\_v\_kube(int \*a) { // в данной версии параметр представляет собой обычный указатель

\*a = \*a \* \*a \* \*a;

return \*a;

}

int chislo\_v\_kube(const int \*a) { // в данной версии параметр представляет собой указатель на константу

return \*a \* \*a \* \*a;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int a = 4;

const int b = 3;

cout << "kube(a): " << chislo\_v\_kube(&a) << ' ' << "a: " << a << endl; // будет использована версия int chislo\_v\_kube(int \*a)

cout << "kube(b): " << chislo\_v\_kube(&b) << ' ' << "b: " << b << endl; // будет использована версия int chislo\_v\_kube(const int \*a)

}

Для большей наглядности в версии, где не константный параметр, мы изменим само число и выведем на экран:



1. **Когда не нужно перегружать функции**

В каких случаях перегрузка имени не дает преимуществ? Например, тогда, когда присвоение функциям разных имен облегчает чтение программы. Рассмотрим такой пример:

double area\_of\_triangle(double a, double h); // функция для подсчёта площади треугольника

double area\_of\_square(double a); // функция для подсчёта площади квадрата

double area\_of\_parallelogram(double a, double b, double s); // функция для подсчёта площади параллелограмма

На первый взгляд, может показаться, что хорошо было бы перегрузить эти функции под общим названием area. Однако этого делать не следует: разные имена несут информацию, без которой программу будет труднее понять.

Применять те или иные возможности следует тогда, когда этого требует логика приложения. Вовсе не обязательно включать перегруженные функции в программу только потому, что они существуют.

1. **Выводы**

Перегрузка – это мощный инструмент, но пользоваться им надо продуманно и аккуратно. В перегрузке немало подводных камней, поэтому надо внимательно смотреть и тщательно всё продумывать, чтобы потом не было ошибок.

Не стоит использовать перегрузку только потому, что компилятор это позволяет. Увлечение перегрузкой может снизить читаемость кода. Во многих случаях лучше дать название, отражающее суть функции.

Нужно избегать использования перегруженных функций, требующих сложных и не до конца понятных алгоритмов разрешения перегрузки.

Не надо объявлять одноименные функции во вложенных областях видимости – это не перегрузка.