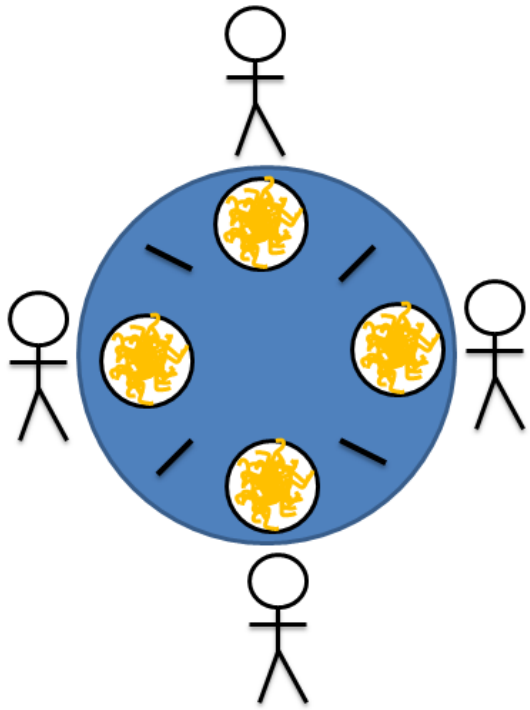
11.misc.md

Задача об обедающих философах

n философов сидят за столом, перед каждым тарелка с едой, между тарелками лежит по 1 палочке. Чтобы поесть, надо взять 2 палочки, то есть философ должен взять палочку слева и справа от своей тарелки.



1. Философы не вежливые

Философ взял правую палочку и не отдает, левую палочку взял сосед и тоже не отдает - произошла взаимная блокировка (deadlock).

2. Философы вежливые

Философ взял правую палочку, посмотрел, что левая занята соседом, положил правую, подумал о вечном, повторил с начала. Остальные поступают аналогично. Если о вечном все думают одинаковое время, то никто из них не сможет поесть (livelock).

3. Философы слишком быстро едят

Поэтому палочка либо слева, либо справа все время занята и философ голодает (starvation).

Решение задачи

1. Иерархия ресурсов

Надо пронумеровать палочки, далее философы берут палочку с наименьшим номером, затем с наибольшим, возращают палочки в обратном порядке. Если n - 1 философов взяли палочке, то останется одна с наибольшим номером, поэтому ее последний философ взять не сможет. Один из взявших палочку берет палочку с наибольшим номером, затем возвращает ее, позволяя поесть следующему.

2. Официант

По запросу выдает палочки или предлагает подождать, если все палочки заняты.

Проблема спящего парикмахера

Цель - парикмахер должен работать когда клиенты есть и спать, когда их нет. Клиент прийдя в парикмахерскую, если парикмахер свободен стрижется, если занят, то идет в приемную и там садится на стул и ждет, если свободных стульев нет, то уходит.

Producer–consumer problem - частный случай этой задачи.

```
struct Client {};

std::mutex barbershop;
std::condition_variable hasClient;

const size_t ChairsNum = 5;

std::queue<Client> clients;

bool clientCame(Client c)
{
    {
        std::unique_lock<std::mutex> lock(barbershop);

        if (clients.size() == ChairsNum)
            return false;

        clients.push(c);
    }

    hasClient.notify_one();

    return true;
}

void barberThread()
{
    while (true)
    {
        Client c;
        {
            std::unique_lock<std::mutex> lock(barbershop);
            while (clients.empty())
            {
                hasClient.wait(lock);
            }
            c = clients.front();
            clients.pop();
        }
        trim(c);
    }
}
```

Задача о читателях-писателях

Есть область памяти, позволяющая чтение и запись. Несколько потоков имеют к ней доступ, при этом одновременно могут читать сколько угодно потоков, но писать — только один. Как обеспечить такой режим доступа?

1. Приоритет читателя

Пока память открыта на чтение, давать читателям беспрепятственный доступ. Писатели могут ждать сколько угодно.

2. Приоритет писателя

Как только появился хоть один писатель, читателей больше не пускать. При этом читатели могут простаивать.

3. Одинаковый приоритет

Независимо от действий других потоков, читатель или писатель должен пройти барьер за конечное время.

Стратегии 1 и 2 чреваты голоданием потоков.

Голодание (starvation) — это более высокоуровневая проблема, чем гонки и взаимоблокировки. Материал для любознательных: стратегии планирования задач

shared_mutex (C++17, boost)

```
#include <boost/thread/shared_mutex.hpp>
#include <boost/thread/locks.hpp>

boost::shared_mutex mutex;

void reader()
{
    boost::shared_lock<boost::shared_mutex> lock(mutex);
    // блокируется если есть unique_lock
    // не блокируется, если есть другие shared_lock
}

void writer()
{
    boost::unique_lock<boost::shared_mutex> lock(mutex);
    // получить эксклюзивный доступ на общих условиях
    // голодания не будет
}

void conditionalWriter()
{
    boost::upgrade_lock<boost::shared_mutex> lock(mutex);
    // блокируется если есть unique_lock
    // не блокируется, если есть другие shared_lock
    if (need_to_write)
    {
        boost::upgrade_to_unique_lock<boost::shared_mutex>
            uniqueLock(lock);
        // получить эксклюзивный доступ
        // блокируется если есть другие shared_lock
        // при высокой конкуренции на чтение может
        // начаться голодание (starvation)
    }
}
```

std::call_once

Проинициализировать что-либо потокобезопасно один раз.

```
std::unique_ptr<Display> display;

void print(const std::string& message)
{
    if (!display)
        display.reset(new Display());
    display->print(message);
}
```

```
std::unique_ptr<Display> display;

static std::once_flag displayInitFlag;

void print(const std::string& message)
{
    std::call_once(displayInitFlag,
        []() { display.reset(new Display()); });
    display->print(message);
}
```

thread_local

Хранилище уровня потока (c++11).

- Создается когда запускается поток
- Уничтожается когда поток завершает работу
- Для каждого потока свое

```
static thread_local std::map<std::string, int> threadCache;
```

Потокобезопасные интерфейсы

```
template <class T>
class queue
{
    bool empty() const;
    T pop();
};

queue<Task> tasks;

if (!tasks.empty())
    process(tasks.pop());
```

```
template <class T>
class ThreadSafeQueue
{
    bool tryPop(T& value)
    {
        std::lock_guard<std::mutex> lock(mutex);
        if (data_.empty())
            return false;
        value = data_.front();
        data_.pop();
        return true;
    }
};

ThreadSafeQueue<Task> tasks;

Task task;
if (tasks.tryPop(task))
        process(task);
```

Асинхронность с колбеками

```
#include <boost/asio.hpp>
```

```
class server
{
    boost::asio::ip::tcp::acceptor acceptor_;
public:
    server(
        boost::asio::io_context& io_context,
        short port)
        : acceptor_(
            io_context,
            boost::asio::ip::tcp::endpoint(
                boost::asio::ip::tcp::v4(), port))
    {
        do_accept();
    }

private:
    void do_accept()
    {
        acceptor_.async_accept(
            [this](
                boost::system::error_code ec,
                boost::asio::ip::tcp::socket socket)
        {
            if (!ec)
            {
                std::make_shared<session>(
                    std::move(socket))->start();
            }

            do_accept();
        });
    }
};
```

```
class session
    : public std::enable_shared_from_this<session>
{
    tcp::socket socket_;
    const size_t max_length = 1024;
    char data_[max_length];
public:
    session(boost::asio::ip::tcp::socket socket)
        : socket_(std::move(socket))
    {
    }

    void start()
    {
        do_read();
    }

private:
    void do_read()
    {
        auto self(shared_from_this());
        socket_.async_read_some(
            boost::asio::buffer(data_, max_length),
            [this, self](
                boost::system::error_code ec,
                std::size_t length)
        {
            if (!ec)
            {
                do_write(length);
            }
        });
    }

    void do_write(std::size_t length)
    {
        auto self(shared_from_this());
        boost::asio::async_write(
            socket_,
            boost::asio::buffer(data_, length),
            [this, self](
                boost::system::error_code ec,
                std::size_t /*length*/)
        {
            if (!ec)
            {
                do_read();
            }
        });
    }
};
```

```
boost::asio::io_context io_context;
server s(io_context, 4000);
io_context.run();
```

Асинхронность с корутинами

```
namespace this_coro =
    boost::asio::experimental::this_coro;

template <typename T>
using awaitable = boost::asio::experimental::awaitable
    <
        T,
        boost::asio::io_context::executor_type
    >;

awaitable<void> listener()
{
    auto executor = co_await this_coro::executor();
```

```
    auto token = co_await this_coro::token();

    boost::asio::ip::tcp::acceptor acceptor(
        executor.context(), { tcp::v4(), 4000 });
    while (true)
    {
        tcp::socket socket =
            co_await acceptor.async_accept(token);
        boost::asio::experimental::co_spawn(executor,
            [socket = std::move(socket)]() mutable
            {
                return echo(std::move(socket));
            },
            boost::asio::experimental::detached);
    }
}
```

```
awaitable<void> echo(tcp::socket socket)
{
    auto token = co_await this_coro::token();

    char data[1024];
    while (true)
    {
        size_t n = co_await socket.async_read_some(
            boost::asio::buffer(data), token);

        co_await boost::asio::async_write(
            socket,
            boost::asio::buffer(data, n),
            token);
    }
}
```

```
boost::asio::io_context io_context;
boost::asio::experimental::co_spawn(
    io_context,
    listener,
    boost::asio::experimental::detached);
io_context.run();
```

Арифметика

Целые отрицательные числа

```
[n][6][5][4][3][2][1][0]
```

Диапазон

Верхняя граница:

2^7 = 128

[0, 127]

Нижняя граница:

-(2^7 + 1)

[-128, 0]

```
  -1 == 0b11111111
-128 == 0b10000000
```

Переполнение

```
uint8_t a = 255;
++a; // overflow
assert(a == 0);
--a; // underflow
assert(a == 255);
```

```
int8_t a = 127;
++a;
// ??? - UB
```

Иллюстрация неопределенного поведения:

```
#include <iostream>

int main()
{
    for(int i = 0; i < 10; ++i)
    {
        auto val = i * 1000000000;
        std::cout << i << ' ' << val << std::endl;
    }
}
```

```
0 0
1 1000000000
2 2000000000
3 -1294967296
4 -294967296
5 705032704
6 1705032704
7 -1589934592
8 -589934592
9 410065408
10 1410065408
11 -1884901888
12 -884901888
...
```

> Проверено на gcc -O2

Приведение знакового к беззнаковому

```
int8_t a = -1;
uint32_t b = a;
assert(b == 4294967295);
```

> Результат копирования знакового бита

Сравнение знакового и беззнакового

```
uint32_t a = 10;
int32_t b = -1;
assert(a < b); // !!!
```

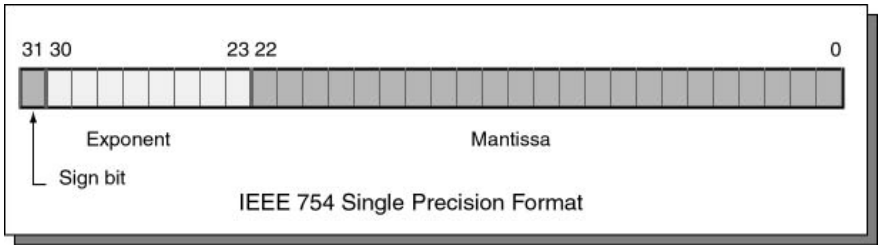
> Если один из аргументов беззнаковый, то он неявно приводится к знаковому.

Плавающая точка

```
float f = 1.234e6;
```

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10^{\overbrace{-4}^{\text{exponent}}}}_{\text{base}}$$

Стандарт IEEE 754



Нормализованная форма

Такая форма, в которой мантисса без учёта знака находится на интервале [1; 10)

```
−158,08 = −0,15808 * 10^3
0,00129 = −0,129 * 10^−2
0,15 = 0,15 * 10^0
```

float

Одинарная точность

```
32 бита
Порядок (exponent) 8 бит
Мантисса (significand) 23 бита
Можно представить целое до 2^24 без потери точности
```

double

Двойная точность

```
64 бита
Порядок (exponent) 11 бит
Мантисса (significand) 52 бита
Можно представить целое до 2^53 без потери точности
```

Специальные значения

Inf (infinity)

Возникает при переполнении или делении не нуля на ноль.

NaN (not a number)

К операциям, приводящим к появлению NaN в качестве ответа, относятся:

- все математические операции, содержащие NaN в качестве одного из операндов;
- деление нуля на ноль;
- деление бесконечности на бесконечность;
- умножение нуля на бесконечность;
- сложение бесконечности с бесконечностью противоположного знака;
- вычисление квадратного корня отрицательного числа;
- логарифмирование отрицательного числа

> NaN не равен ни одному другому значению (даже самому себе); соответственно, самый простой метод проверки результата на NaN — это сравнение полученной величины с самой собой.

Ноль

Так как в нормализованной форме ноль невозможно представить единственным способом.

Сравнение чисел с плавающей точкой на равенство

```
double a = 10 / 3.;
double b = a;
const bool isEqual = a == b; // Ok
```

```
double a = ...;
double b = ...;
// const bool isEqual = a == b; ???
const bool isEqual = fabs(a - b) < 0.0001;
```

> Погрешность представления числа увеличивается с ростом самого этого числа

https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/

Арифметика с массивами чисел с плавающей точкой

> При сложении большого и малого чисел малое просто исчезнет

1. Сортируем массив по возрастанию
2. Получаем сумму от 0 до максимального положительного
3. Получаем сумму от 0 до минимального отрицательного
4. Складываем результаты 2 и 3

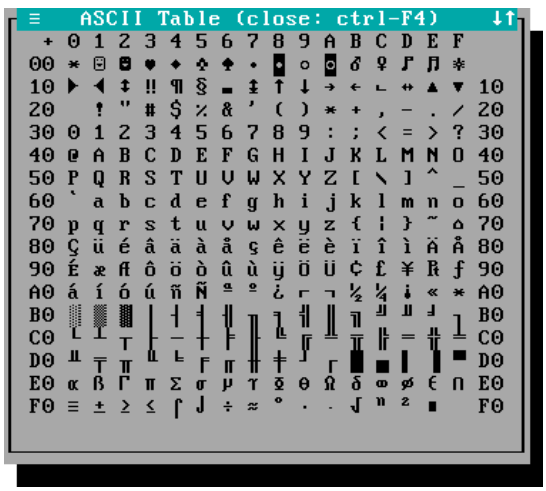
Unicode

ASCII

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

ASCII 8



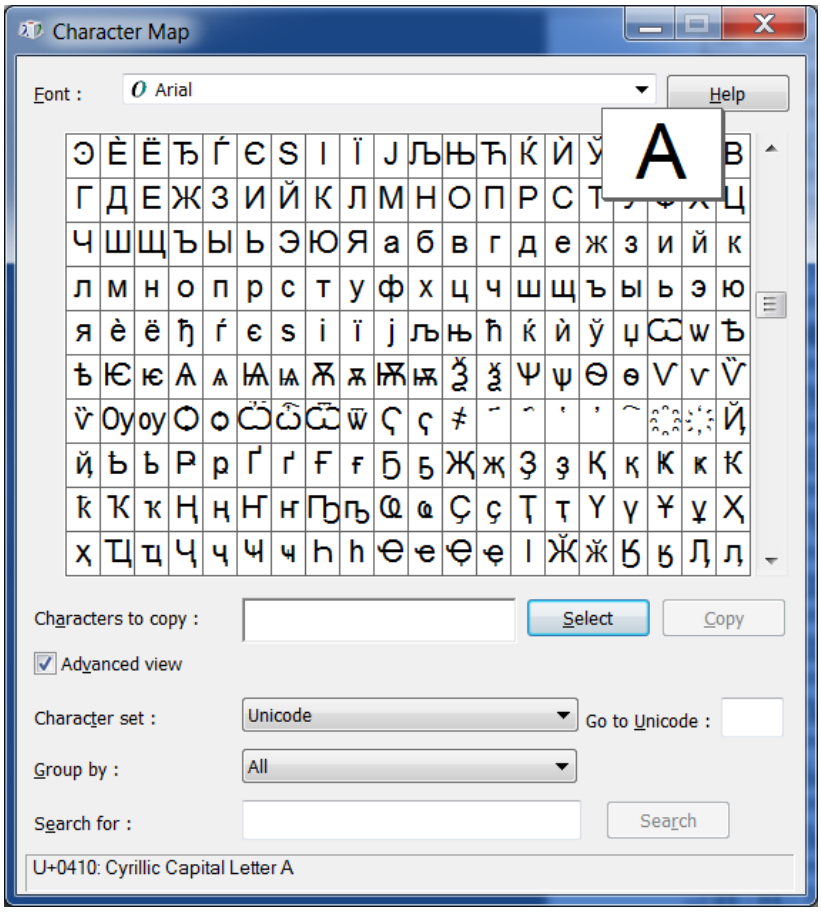
KOI8-R

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	─	│	┌	┐	└	┘	├	┤	┬	┴	┼	▀	▄	▌	▐	▀
9	░	▒	▓	⌠	■	·	√	≈	≤	≥		⌡	°	²	·	÷
A	═	║	╒	ё	╓	╔	╕	╖	╗	╘	╙	╚	╛	╜	╝	╞
B	╟	╠	╡	Ё	╢	╣	╤	╥	╦	╧	╨	╩	╪	╫	╬	n
C	ю	а	б	ц	д	е	ф	г	х	и	й	к	л	м	н	о
D	п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
E	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О
F	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	З	Ш	Э	Щ	Ч	Ъ

Множество кодировок

- Unicode – стандарт, а не кодировка
- CodePoint – символ в таблице Unicode (U+xxxx, где xxxx шестнадцетеричное число, например, U+0410)
- На данный момент уже существует 9 версия стандарта
- Начиная с версии 2.0 кодовая область расширена за пределы 2^16
- В версии 6.0 описано 109 242 графических и 273 прочих символа

http://www.unicode.org/



UCS-2

```
H      E      L      L      O
U+0048 U+0065 U+006C U+006C U+006F
```

```
00 48 00 65 00 6C 00 6C 00 6F (Big Endian)
    или
48 00 65 00 6C 00 6C 00 6F 00 (Little Endian)
```

BOM (byte order mark)

UTF-8

```
EF BB BF
```

UTF-16

```
FE FF (Big Endian)
FF FE (Little Endian)
```

UTF-32

```
00 00 FE FF (Big Endian)
FF FE 00 00 (Little Endian)
```

UTF-16

- Позволяет отобразить 2^20 + 2^16 – 2048 (1 112 064) символов
- Кодирует 0000..D7FF и E000..10FFFF (в виде суррогатных пар)
- Исключенный диапазон D800..DFFF используется как раз для кодирования суррогатных пар
- Суррогатная пара – символ кодируемый двумя 16 битными словами

UTF-8

Переменное количество байт от 1 до 6 Позволяет закодировать 2^31 CodePoints

Диапазон символов	Количество байт
00000000-0000007F	1
00000080-000007FF	2
00000800-0000FFFF	3
00010000-001FFFFF	4
00200000-03FFFFFF	5
04000000-7FFFFFFF	6

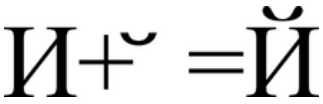
UTF-32, UCS-4

- Кодирует любой символ 4 байтами
- Позволяет индексировать 231 CodePoints
- Символы Юникод непосредственно индексируемы

Но и это еще не все

Модифицирующие символы

Графические символы в Юникоде подразделяются на протяжённые и непротяжённые (бесширинные). Непротяжённые символы при отображении не занимают места в строке. К ним относятся, в частности, знаки ударения и прочие диакритические знаки. Как протяжённые, так и непротяжённые символы имеют собственные коды. Протяжённые символы иначе называются базовыми (base characters), а непротяжённые — модифицирующими (combining characters); причём последние не могут встречаться самостоятельно. Например, символ «á» может быть представлен как последовательность базового символа «a» (U+0061) и модифицирующего символа « ́» (U+0301) или как монолитный символ «á» (U+00E1).



Алгоритмы нормализации

Поскольку одни и те же символы можно представить различными кодами, сравнение строк байт за байтом становится невозможным. Алгоритмы нормализации (normalization forms) решают эту проблему, выполняя приведение текста к определённому стандартному виду. Приведение осуществляется путём замены символов на эквивалентные с использованием таблиц и правил. «Декомпозицией» называется замена (разложение) одного символа на несколько составляющих символов, а «композицией», наоборот, — замена (соединение) нескольких составляющих символов на один символ.

В стандарте Юникода определены 4 алгоритма нормализации текста: NFD, NFC, NFKD и NFKC.

Что еще?

Например, двунаправленное письмо.

Стандарт Юникод поддерживает письменности языков как с направлением написания слева направо (left-to-right, LTR), так и с написанием справа налево (right-to-left, RTL) – например, арабское и еврейское письмо. В обоих случаях символы хранятся в «естественном» порядке; их отображение с учётом нужного направления письма обеспечивается приложением.

Кроме того, Юникод поддерживает комбинированные тексты, сочетающие фрагменты с разным направлением письма.

И многое другое...

ICU

http://site.icu-project.org/

Инструменты

Valgrind

Заменяет стандартное выделение памяти своей реализацией отслеживающей корректное ее использование и освобождение.

Позволяет определить:

- Попытки использования неинициализированной памяти
- Чтение/запись в память после её освобождения
- Чтение/запись за границами выделенного блока
- Утечки памяти

Статический анализ. CppCheck

Используется для статического анализа кода. Возможности:

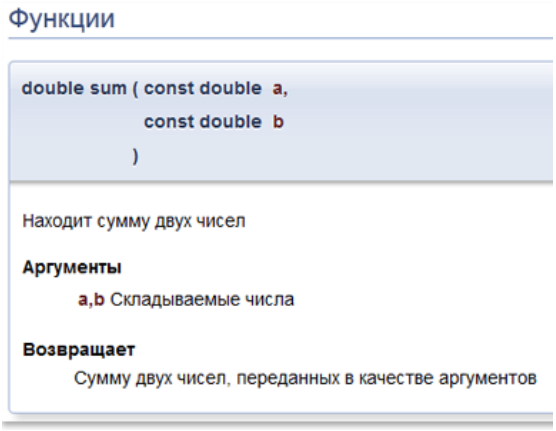
1. Проверяет выход за пределы
2. Обнаруживает утечки памяти
3. Обнаруживает возможное разыменовывание NULL-указателей
4. Обнаруживает неинициализированные переменные
5. Обнаруживает неправильное использование STL
6. Проверяет обработку исключительных ситуаций на безопасность
7. Находит устаревшие и неиспользуемые функции
8. Предупреждает о неиспользуемом или бесполезном коде
9. Находит подозрительные участки кода, которые могут содержать в себе ошибки

```
[check.cpp:11]: (error) Array 'c[10]' index 10 out of bounds
[check.cpp:5]: (error) Memory leak: __p
[check.cpp:17]: (error) Memory leak: a
[check.cpp:14]: (error) Mismatching allocation and deallocation: A::__p
[check.cpp:8]: (error) Null pointer dereference
```

Doxygen

Документацию можно писать в коде. Doxygen проанализирует исходный код, построит документацию (html, latex и т.д.) из определений типов и дополнит ее найденными комментариями записанными в специальном формате:

```
/*!
Находит сумму двух чисел
@param a,b Складываемые числа
@return Сумму двух чисел, переданных в качестве аргументов
*/
double sum(const double a, const double b);
```



Функции

double sum (const double a,
const double b
)

Находит сумму двух чисел

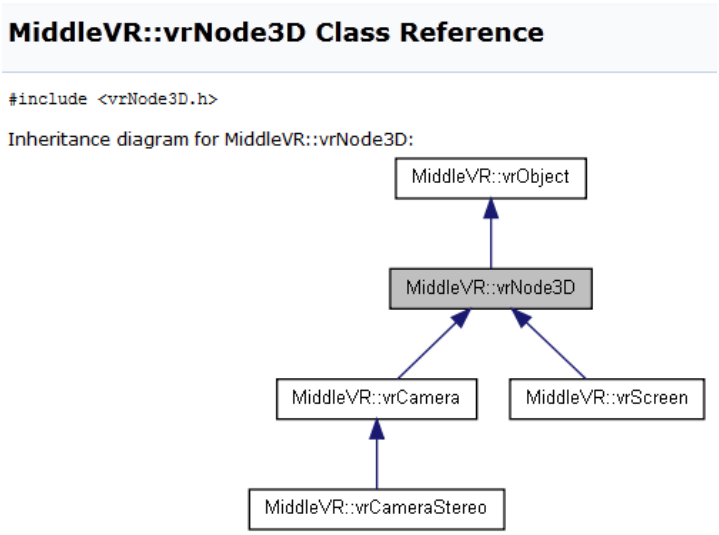
Аргументы

a,b Складываемые числа

Возвращает

Сумму двух чисел, переданных в качестве аргументов

Если установлен Graphviz можно дополнить документацию графом вызовов и диаграммами наследования.



Ускорение сборки

В разных единицах трансляции могут использоваться одни и те же заголовочные файлы – это приводит к многократной компиляции одного и того же кода.

Напрашивается идея собрать общие заголовочные файлы в один и скомпилировать их один раз – это предварительно откомпилированные заголовочные файлы (precompiled headers):

stdafx.h

```
#include <vector>
```

a.cpp

```
#include"stdafx.h" // Всегда первой строкой
#include "a.h"

...
```

При изменении stdafx.h или включенного в него заголовочного файла перекомпилируется весь проект!

EOF