

Структура языка программирования. Стандартная библиотека

1. Стандартная библиотека:

В составе языка программирования, как правило, есть обязательный (стандартный) набор функций. Такие функции называют встроенными.

Встраиваться функции могут тремя способами:

- прямо в код транслятора;
- находиться в отдельной библиотеке;
- сочетание первого и второго случаев.

Основные требования к набору средств стандартной библиотеки (Бьёрн Страуструп):

- *эффективность*;
- *независимость от алгоритмов* — должна предоставлять возможность задавать алгоритмы в качестве параметров;
- *удобство и безопасность*;
- *завершённость*;
- *органично сочетаться с языком*;
- *типобезопасность*;
- *поддержка общепринятых стилей программирования*;
- *расширяемость* — способность единообразно работать со встроенными типами данных и с типами, определяемыми пользователем

Подходы к разработке стандартных библиотек языков программирования:

- должна содержать в себе только те процедуры и функции, которые используются практически всеми и обладают максимальной универсальностью;
- должна содержать в себе максимально возможное количество типичных алгоритмов, обеспечивать простую работу с большинством объектов (в идеале, со всеми), с которыми может взаимодействовать программа. Пример реализации этого подхода является язык Python с девизом «Batteries included» (батарейки в комплекте).

Второй подход основывается на утверждении, что скорость написания программ и их корректность важнее эффективности. Программист должен иметь максимум готовых, проверенных библиотечных функций, которые он сможет использовать.

Программирование «вручную» необходимо только для нетривиальных алгоритмов.

Результат: экономия времени и минимизация технических ошибок при написании кода.

Состав.

В зависимости от возможностей языка, стандартная библиотека может содержать:

- процедуры и функции
- макросы
- глобальные переменные
- классы
- шаблоны

2. Стандартная библиотека C++: ANSI C и STL.

- ✓ имя и характеристики каждой функции указываются в заголовочном файле;
- ✓ текущая реализация функций описана отдельно в библиотечном файле.
- ✓ Стандартная библиотека обычно поставляется вместе с компилятором.

Стандартные библиотеки C и C++

Ни в C, ни в C++ нет ключевых слов, обеспечивающих ввод-вывод, обрабатывающих строки, выполняющих различные математические вычисления или какие-нибудь другие полезные процедуры. Все эти операции выполняются за счет использования набора библиотечных функций, поддерживаемых компилятором. Существует два основных вида библиотек: библиотека C-функций, которая поддерживается всеми компиляторами C и C++, и библиотека классов C++, которая, разумеется, годится только для языка C++.

Язык программирования C++ содержит два вида библиотек.

В первой библиотеке хранятся стандартные универсальные функции, не принадлежащие ни одному классу, которая *унаследована от языка C*.

Вторая библиотека содержит библиотеку классов и является *объектно-ориентированной*.

3. Стандартная библиотека C++: ANSI C89

(заголовки <сxxxx>, пространство имен std).

В 1983 году Американским национальным институтом стандартов (ANSI) был сформирован комитет для разработки стандарта языка Си.

В 1989 году принят стандарт **C89**, в который был включен набор библиотек, названный **Стандартная библиотека ANSI Си**.

Стандартная Библиотека современного языка C++ включает в себя спецификации стандарта **ISO C90** стандартной библиотеки языка Си и представляет собой набор файлов заголовков.

В новых файлах заголовков отсутствует расширение .h.

Каждый заголовочный файл из стандартной библиотеки языка Си включен в стандартную библиотеку языка C++ под именами, созданными путём отсечения расширения .h и добавлением 'c' в начале.

Пример, 'time.h' стал 'ctime'.

Единственное отличие между этими файлами заголовков и традиционными заголовочными файлами стандартной библиотеки языка Си заключается в том, что функции должны быть помещены в пространство имен **std::**

4. Стандартная библиотека C++: STL

Стандартная библиотека шаблонов работает на большинстве комбинаций платформ/компиляторов, включая cfront, Borland, Visual C++, Set C++, ObjectCenter (UNIX C/C++) и последние компиляторы от Sun&HP.

Авторы Стандартной библиотеки шаблонов – наш соотечественник Алекс Степанов и Менг Ли.



В интервью Степанов сказал:

«Я начал размышлять об обобщённом программировании в конце 70-х, когда заметил, что некоторые алгоритмы зависят не от конкретной реализации структуры данных, а лишь от небольшого числа существенных семантических свойств этой структуры. Так что я начал рассматривать самые разные алгоритмы, и обнаружил, что большинство из них могут быть абстрагированы от конкретной реализации так, что эффективность при этом не теряется. Эффективность является для меня одной из

основных забот».

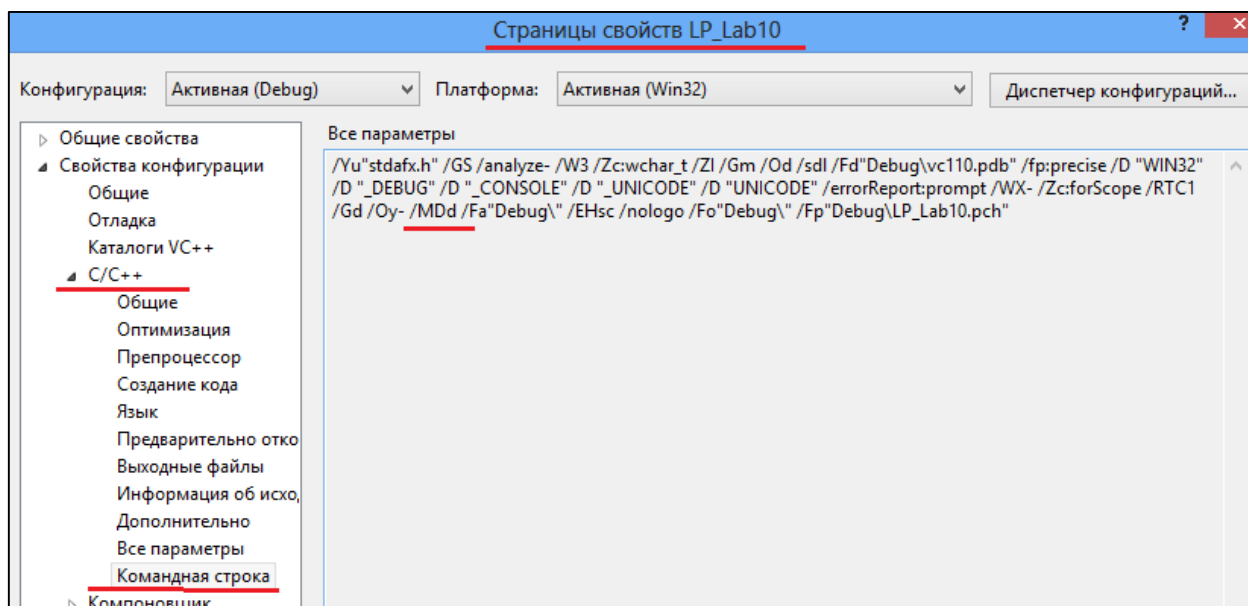
5. ANSI C + STL

Большинство библиотек поддерживает как *статическое связывание* (для связывания библиотеки непосредственно в коде), так и *динамическое связывание* (для использования в коде общих библиотек DLL).

Таблица ниже содержит LIB-файлы, входящие в библиотеки времени выполнения, а также связанные с ними параметры компилятора и директивы препроцессора:

Стандартная библиотека C++			
Стандартная библиотека C++	Характеристики	Параметр	Директивы препроцессора
LIBCPMT.LIB	Многопоточный, статические связи	/MT	_MT
MSVCPRT.LIB	Многопоточный, динамические ссылки (библиотека импорта MSVCP110.dll)	/MD	_MT, _DLL
LIBCPMTD.LIB	Многопоточный, статические связи	/MTd	_DEBUG, _MT
MSVCPRTD.LIB	Многопоточный, динамические ссылки (библиотека импорта MSVCP110D.БИБЛИОТЕКА DLL)	/MDd	_DEBUG, _MT, _DLL

При сборке финальной версии проекта одна из базовых библиотек времени выполнения C (libcmtd.lib, msvcmrtd.lib, msvcrtd.lib) из таблицы будет скомпонована по умолчанию, *в зависимости* от выбранного параметра компилятора.



При включении в код любого из файлов заголовков стандартной библиотеки C++, Visual C++ автоматически подключит во время компиляции стандартную библиотеку C++.

Пример 1: (C++)

```
#include <ios>
```

Пример 2: (<cstring>)

```
#include "stdafx.h"
#include <cstring>

int _tmain(int argc, _TCHAR* argv[])
{
    int k = strlen("XXXXXXXXXXx");
    return 0;
}
```

```
#pragma once
#ifndef _CSTRING_
#define _CSTRING_
#include <yvals.h>

#ifdef _STD_USING
#undef _STD_USING
#include <string.h>
#define _STD_USING
#else /* _STD_USING */
#include <string.h>
#endif /* _STD_USING */

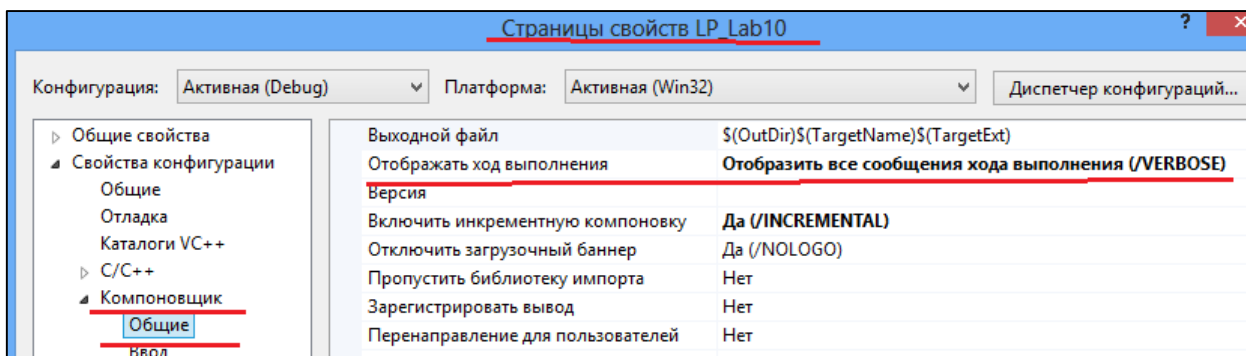
#if _GLOBAL_USING && !defined(RC_INVOKED)
_STD_BEGIN
using _CSTD size_t; using _CSTD memchr; using _CSTD memcmp;

using _CSTD memcpy; using _CSTD memmove; using _CSTD memset;
using _CSTD strcat; using _CSTD strchr; using _CSTD strcmp;
using _CSTD strcoll; using _CSTD strcpy; using _CSTD strcspn;
using _CSTD strerror; using _CSTD strlen; using _CSTD strncat;
using _CSTD strncmp; using _CSTD strncpy; using _CSTD strpbrk;
using _CSTD strrchr; using _CSTD strspn; using _CSTD strstr;
using _CSTD strtok; using _CSTD strxfrm;
_STD_END
#endif /* _GLOBAL_USING */
#endif /* _CSTRING_ */
```

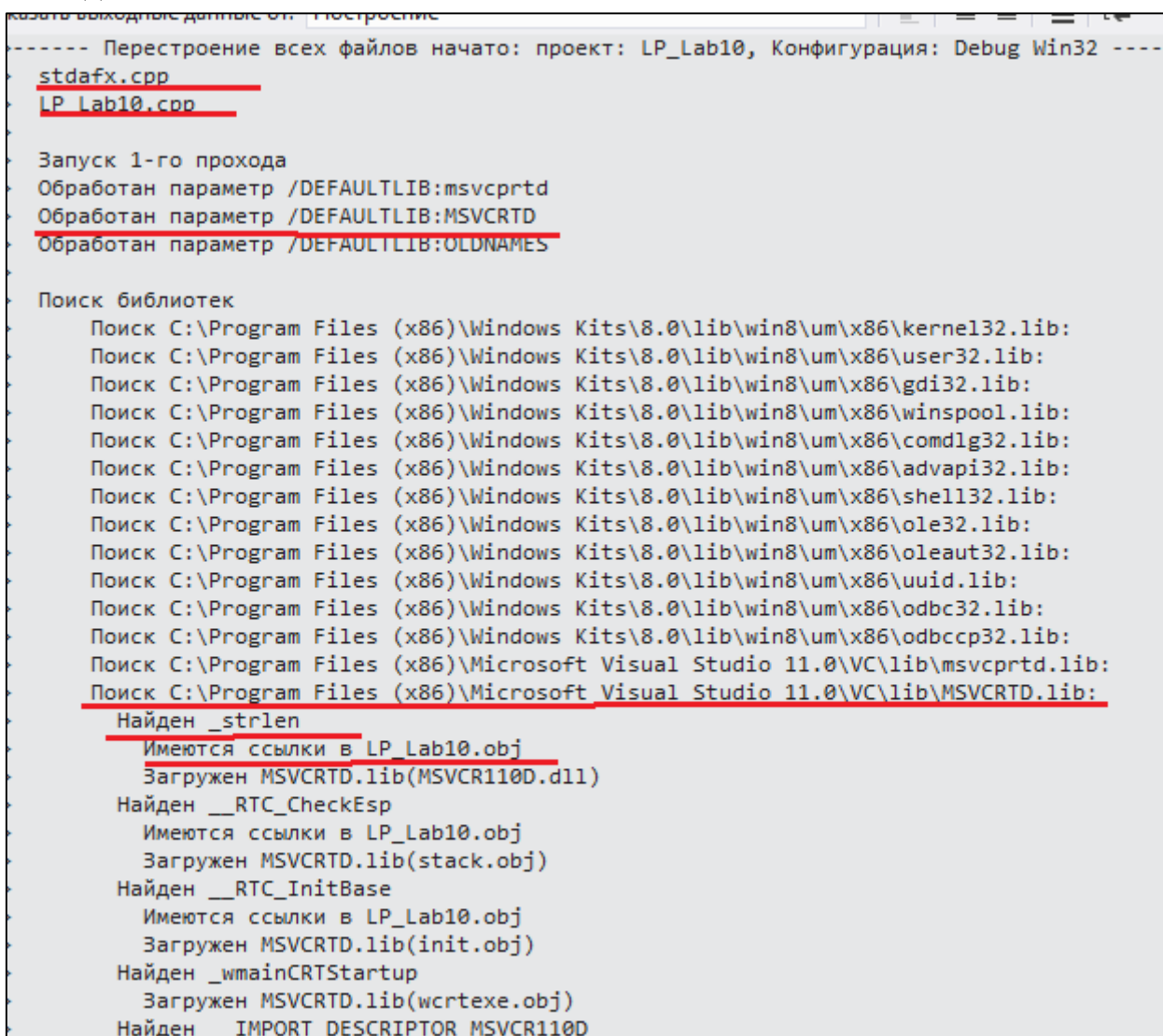
Заголовочный файл <string.h>

```
_DEFINE_CPP_OVERLOAD_STANDARD_FUNC_0_1(char *, __RETURN_POLICY_DST, __EMPTY_DECLSPEC, strcpy,
#if __STDC_WANT_SECURE_LIB__
_Check_return_wat_ _CRTIMP_ALTERNATIVE errno_t __cdecl strcat_s(_Inout_updates_z_(_SizeInBytes)
#endif
_DEFINE_CPP_OVERLOAD_SECURE_FUNC_0_1(errno_t, strcat_s, char, _Dest, _In_z_ const char *, _Sou
_DEFINE_CPP_OVERLOAD_STANDARD_FUNC_0_1(char *, __RETURN_POLICY_DST, __EMPTY_DECLSPEC, strcat,
    _Check_return_ int __cdecl strcmp(_In_z_ const char * _Str1, _In_z_ const char * _S
    _Check_return_ size_t __cdecl strlen(_In_z_ const char * _Str);
_Check_return_ _CRTIMP
_When_(_MaxCount > _String_length_(_Str), _Post_satisfies_(return == _String_length_(_Str)))
_When_(_MaxCount <= _String_length_(_Str), _Post_satisfies_(return == _MaxCount))
size_t __cdecl strlen(_In_reads_or_z_(_MaxCount) const char * _Str, _In_ size_t _MaxCount);
#if __STDC_WANT_SECURE_LIB__ && !defined(__midl)
```

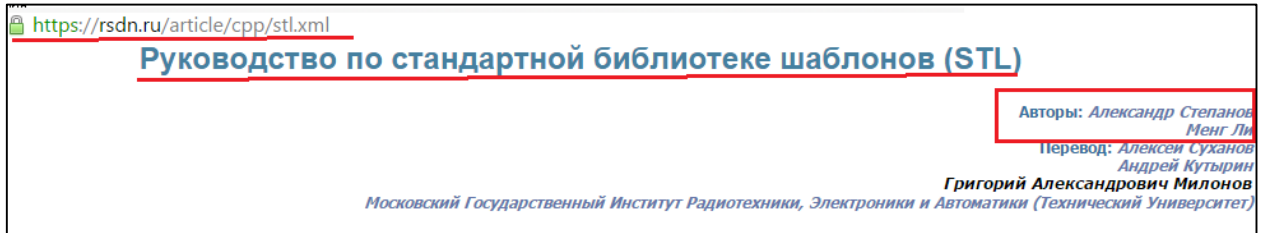
Можно проследить за ходом выполнения, используя ключ командной строки /VERBOSE



Вывод:



6. Стандартная библиотека C++ STL:



7. Стандартная библиотека C++ STL:

библиотека стандартных шаблонов (**STL**) – набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

STL (Standard Template Library) – стандартная библиотека шаблонов. Библиотека содержит универсальные шаблонные классы и функции, реализующие большое количество распространённых универсальных алгоритмов и структур данных. Т.к. библиотека **STL** состоит из шаблонных классов, ее алгоритмы и структуры можно применять практически к любым типам данных.

Частью *стандартной библиотеки C++* является библиотека **STL**. Библиотека STL содержит пять основных видов компонентов:

- **контейнер** (*container*): управляет набором объектов в памяти.
- **итератор** (*iterator*): обеспечивает для алгоритма средство доступа к содержимому контейнера.
- **алгоритм** (*algorithm*): определяет вычислительную процедуру.
- **функциональный объект** (*function object*): инкапсулирует функцию в объекте для использования другими компонентами.
- **адаптер** (*adaptor*): адаптирует компонент для обеспечения различного интерфейса.

Контейнер – объект, содержащий другие объекты (стек, список, очередь, вектор и пр.), предназначенный для хранения однотипных объектов и обеспечения доступа к ним.

Контейнеры:

последовательные (последовательный доступ к элементам);
ассоциативные (доступ по ключу).

В библиотеке STL реализованы следующие контейнеры:

- **bitset** – набор битов;
- **deque** – двусторонняя очередь;
- **list** – линейный список;
- **map** – ассоциативный контейнер, построенный по принципу key:value (ключ:значение), в котором каждому ключу key соответствует значение value (пары хранятся в отсортированном виде, что позволяет осуществлять быстрый поиск по ключу);
- **multimap** – ассоциативный контейнер, в котором одному значению (key) соответствует несколько значений (value1, value2, ..., valueN);
- **multiset** – множество, в котором один и тот же элемент может встречаться несколько раз;
- **priority_queue** – очередь с приоритетами;
- **queue** – очередь;
- **set** – множество, в котором каждый элемент встречается только один раз;
- **stack** – стек;
- **vector** – динамический массив (коллекция элементов, сохраненных в массиве, размер может изменяться по мере необходимости).

Алгоритм определяет вычислительную процедуру (обобщённые алгоритмы) для работы с контейнерами. Алгоритмы позволяют манипулировать содержимым контейнера: инициализировать, сортировать, искать, изменять содержимое контейнера.

Содержимое контейнеров обрабатывается с помощью алгоритмов. Алгоритмы позволяют обрабатывать контейнеры на любой вкус: инициализировать, сортировать содержимое контейнеров, преобразовывать, реализовывать различные виды поиска и тому подобное.

Для доступа к алгоритмам, нужно подключить соответствующую библиотеку `#include <algorithm>`

Все реализованные функции можно поделить на три группы:

- Методы перебора всех элементов коллекции и их обработки: `count`, `count_if`, `find`, `find_if`, `adjacent_find`, `for_each`, `mismatch`, `equal`, `search` `copy`, `copy_backward`, `swap`, `iter_swap`, `swap_ranges`, `fill`, `fill_n`, `generate`, `generate_n`, `replace`, `replace_if`, `transform`, `remove`,

- remove_if, remove_copy, remove_copy_if, unique, unique_copy, reverse, reverse_copy, rotate, rotate_copy, random_shuffle, partition, stable_partition
- Методы сортировки коллекции:
 - sort, stable_sort, partial_sort, partial_sort_copy, nth_element, binary_search, lower_bound, upper_bound, equal_range, merge, inplace_merge, includes, set_union, set_intersection, set_difference, set_symmetric_difference, make_heap, push_heap, pop_heap, sort_heap, min, max, min_element, max_element, lexicographical_compare, next_permutation, prev_permutation
- Методы выполнения определенных арифметических операций над членами коллекций:
 - Accumulate, inner_product, partial_sum, adjacent_difference

Все алгоритмы являются шаблонными функциями. Они могут быть применены к любому типу контейнера. Ниже приведен сокращенный перечень алгоритмов библиотеки STL:

- **count** — подсчитывает количество вхождений заданного элемента в последовательности;
- **count_if** — вычисляет количество вхождений элемента в последовательности, соответствующей заданному условию;
- **equal** — определяет, совпадают ли элементы двух диапазонов;
- **fill, fill_n** — заполняют диапазон нужными значениями;
- **find** — осуществляет поиск элемента в заданной последовательности, возвращает позицию (итератор) первого вхождения элемента в последовательности;
- **find_if** — находит первый элемент последовательности, который совпадает с элементом с другой последовательности;
- **for_each** — применяет указанную функцию к заданному диапазону элементов;
- **max** — возвращает максимум из двух значений;
- **min** — из двух значений возвращает минимальное;
- **replace** — заменяет элементы заданного диапазона из одного значения на другое;
- **search** — выполняет поиск подпоследовательности в последовательности;
- **sort** — упорядочивает последовательность в заданном диапазоне;
- **swap** — меняет местами значения, заданные ссылками;

Итератор: объект, обеспечивающий для алгоритма средство доступа к содержимому контейнера. Итератор позволяет перемещаться по

содержимому контейнера подобно тому, как указатель перемещается по элементам массива.

Итераторы:

произвольного доступа (для ассоциативных контейнеров);
двунаправленные (для последовательных).

Итераторы — это объекты, которые позволяют перемещаться по содержимому контейнера. Итераторы подобны указателям. Итераторы являются абстракцией указателя. С помощью итераторов можно считывать и изменять значения элементов контейнера.

Чтобы использовать итераторы нужно подключить заголовок:

```
#include <iterator>
```

Существуют три типа итераторов:

- (forward) iterator — для обхода коллекции от меньшего индекса к большему;
- reverse iterator — для обхода коллекции от большего индекса к меньшему;
- random access iterator — для обхода коллекции в любом направлении.

Функциональный объект. Для обеспечения гибкого взаимодействия между итераторами, контейнерами и алгоритмами, используются функторы.

Функтор — это класс, в котором реализована *операторная* функция `operator()`. Благодаря функторам объект класса представляется как функция (часто представляется как лямбда-функция).

Чтобы использовать функторы в программе, нужно подключить заголовок `functional`

```
#include <functional>
```

В библиотеке STL функторы делятся на две категории:

- бинарные — содержат два аргумента;
- унарные — содержат один аргумент.

Библиотека STL содержит следующие бинарные функторы:

- `plus` — суммирует (+) два аргумента;
- `minus` — вычитает (−) аргументы;
- `multiplies` — умножает (*) два аргумента;
- `divides` — делит (/) аргументы;
- `modulus` — возвращает результат операции % для двух аргументов;

- `equal_to` — сравнивает два аргументы на равенство (`==`);
- `not_equal_to` — сравнивает два аргументы на неравенство (`!=`);
- `greater` — определяет, больше ли первый аргумент чем второй аргумент (`>`);
- `greater_equal` — определяет, первый аргумент есть больше или равен второму аргументу (`>=`);
- `less` — определяет, меньше ли первый аргумент чем второй аргумент (`<`);
- `less_equal` — определяет, первый аргумент меньше или равен второму аргументу (`<=`);
- `logical_and` — применяет к аргументам логическое «И» (AND);
- `logical_or` — применяет к двум аргументам логическое «ИЛИ» (OR).

Также определены два унарных функтора:

- `logical_not` — применяет к аргументу логическое «НЕТ» (NOT);
- `negate` — изменяет знак своего аргумента на противоположный.