Лекция 10
БГТУ, ФИТ, ПОИТ, 2 семестр, Языки программирования

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.

Язык программирования C++ содержит два вида библиотек.

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

Вторая библиотека содержит библиотеку классов и является объектно-ориентированной.

3. **Стандартная библиотека C++: ANSI C89** (заголовки <cxxxx>, 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

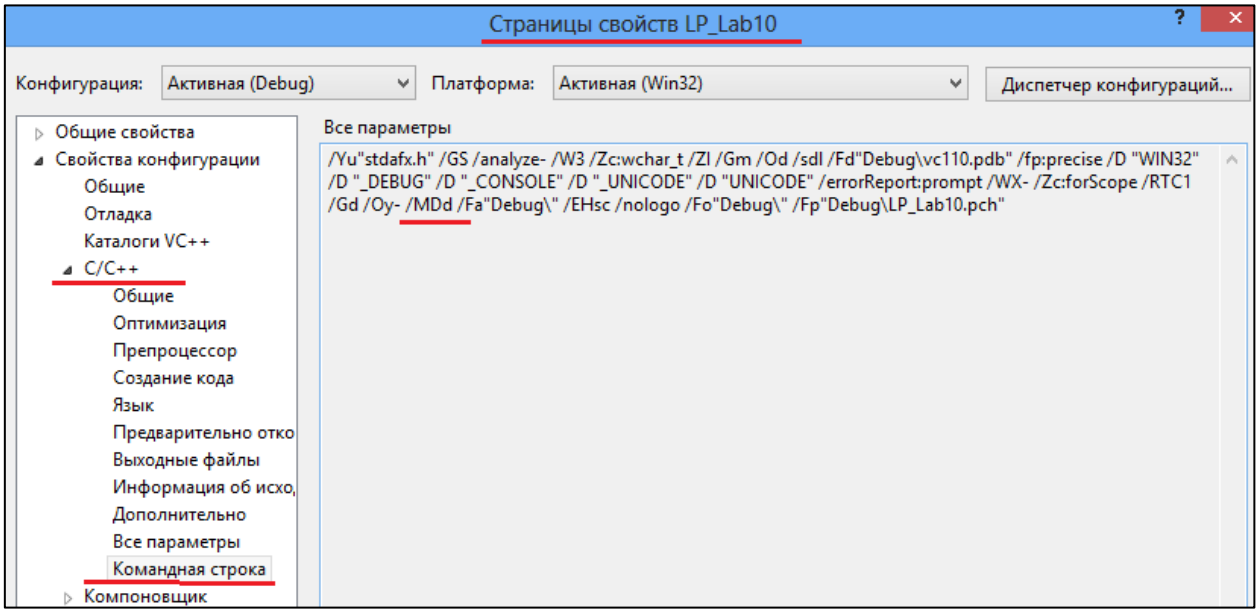
Большинство библиотек поддерживает как статическое связывание (для связывания библиотеки непосредственно в коде), так и динамическое связывание (для использования в коде общих библиотек DLL).

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 (libcmt.lib, msvcmrt.lib, msvcrt.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:

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

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

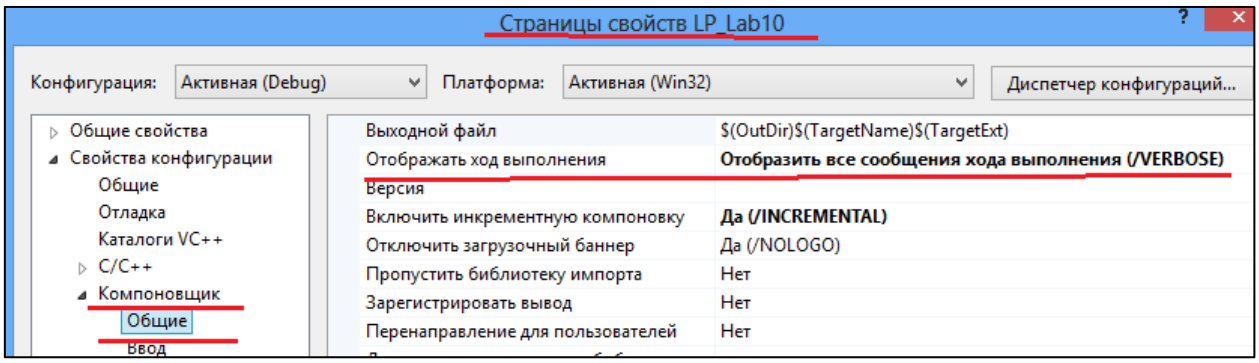
<cstring>

```
#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_ */
```

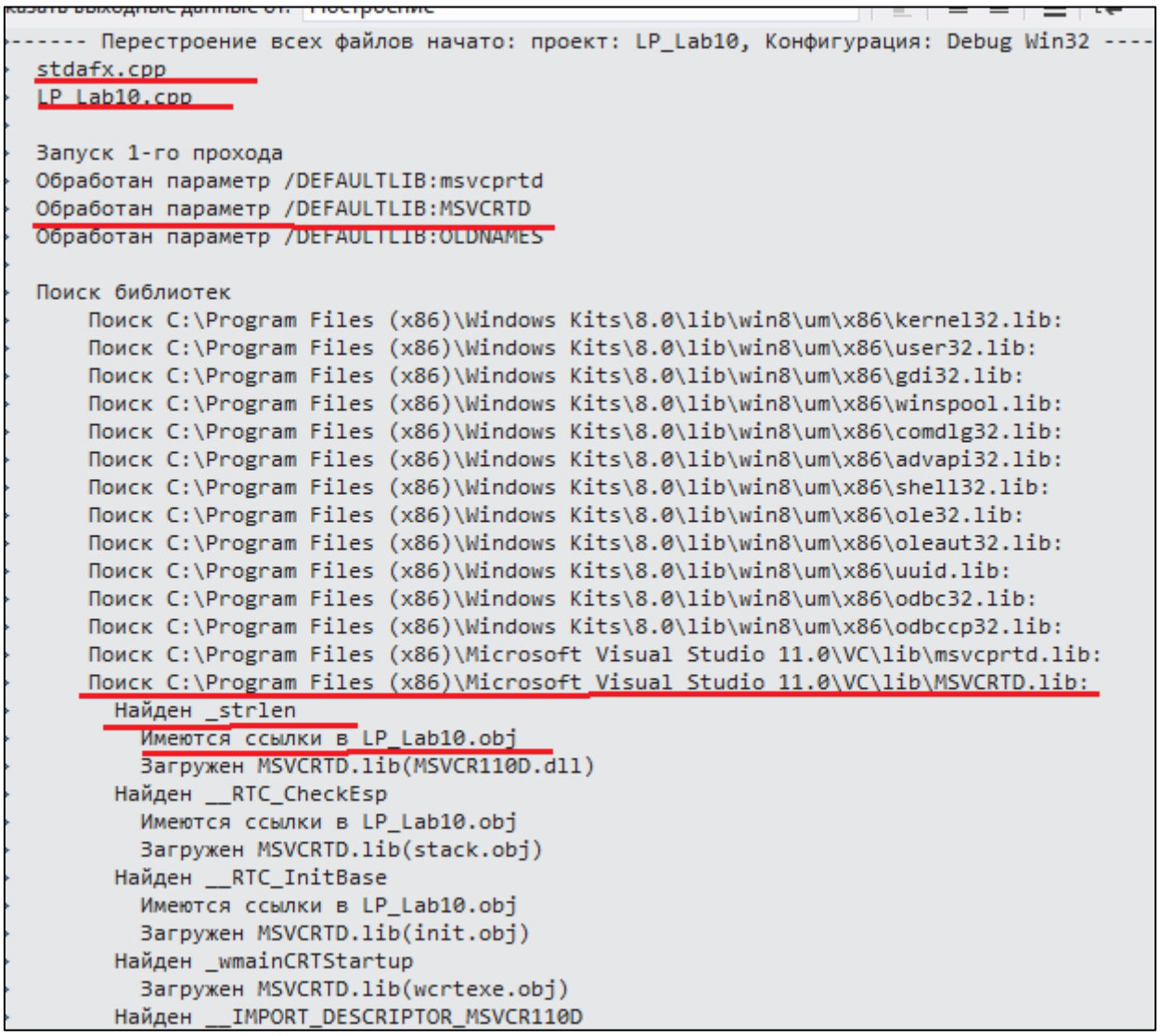
Заголовочный файл **<string.h>**

```
__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 strnlen(_In_reads_or_z_(_MaxCount) const char * _Str, _In_ size_t _MaxCount);
#if __STDC_WANT_SECURE_LIB__ && !defined (__midl)
```

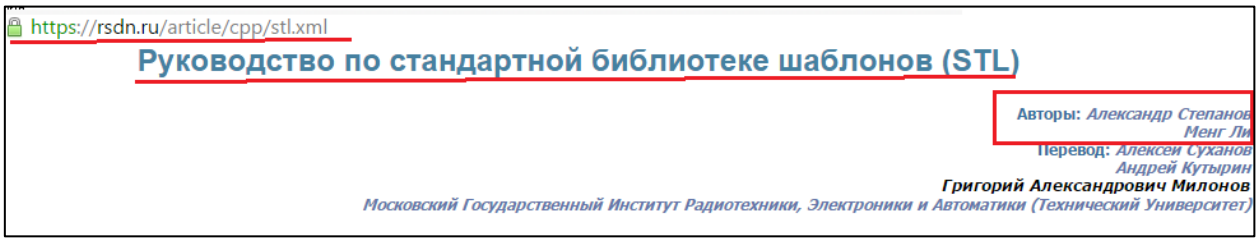
Можно проследить за ходом выполнения, используя ключ компановщика /VERBOSE



Вывод:



6. **Стандартная библиотека C++ STL:**



7. **Стандартная библиотека C++ STL:**

STL — стандартная библиотека шаблонов. Библиотека содержит универсальные шаблонные классы и функции, реализующие большое количество распространенных универсальных алгоритмов и структур данных. Т.к. библиотека **STL** состоит из шаблонных классов, ее алгоритмы и структуры можно применять практически к любым типам данных.

Основные компоненты библиотеки STL:

1) контейнеры + итераторы;
2) алгоритмы;
3) потоки ввода-вывода.

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

Например:
класс **vector** – определяет динамический массив;
класс **stack** – определяет стек;
класс **list** – позволяет работать с линейным списком;
класс **deque** – создает двухстороннюю очередь.

Контейнеры:

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

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

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).