

План лекции

Общие сведения

Библиотека libc

Библиотека glibc

Оглавление руководства по библиотеке glibc

Текст подключаемый препроцессором C к файлу call-as.c

Общие сведения

Язык C спроектирован как лаконичный язык и не предоставляет встроенных средств на уровне его ключевых слов для выполнения таких распространенных операций, как:

- низкоуровневой или форматированный ввод/вывод;
- управление строками;
- вычисление арифметических и математических функций;
- управление памятью, процессами, файловыми системами, сетевыми соединениями;
- и т. п.

Для решения таких задач применяется стандартная библиотека языка C (также известная как libc). Сама по себе она не является частью языка, однако предусмотренный в ней набор функций, а также определений типов и макроопределений составляет системную среду, поддерживающую разработку ПО на C.

В 1978 году **Брайан Керниган** и **Деннис Ритчи** опубликовали первую редакцию книги «Язык программирования C», содержащую систематическое описание такого подхода. Эта книга, известная среди программистов как «K&R», служила многие годы неформальной спецификацией языка и выдержала в США около сорока изданий. Подход «K&R» и его реализации оказались очень плодотворными и активно использовались при разработке большого числа проектов.

Язык C был и остаётся одним из самых распространённых языков программирования в течение пятидесяти лет и, по сути является - языка общения (*lingua franca*) между разработчиками. Большая группа языков унаследовали базовый синтаксис C (использование фигурных скобок в качестве ограничителей блоков кода, описание переменных, характерные формы операторов `for`, `while`, `if`, `switch` с параметрами в скобках, комбинированные операции `++`, `--`, `+=`, `-=` и другие), из-за чего программы на этих языках имеют характерный внешний вид, ассоциирующийся именно с C. Это такие языки как C++, Objective C, Java, JavaScript, PHP, Perl, AWK, C#, Go и др.

C — язык системного программирования:

- Эффективный машинный код. Используется вместо ассемблера.
- Применяется при разработке системного ПО для платформ от супер ЭВМ до встроенных.
- Неполный список продуктов, реализованных на C.
 - ◆ Ядра ОС Unix (и всех ее производных), MS Windows, Linux, OS X, iOS, Android, Windows Phone.
 - ◆ СУБД Oracle Database, MySQL, MS SQL Server, PostgreSQL.

- ♦ Первые реализации языков Java, Python, Perl и PHP.
- Cpython - эталонная реализация языка Python.
- Промежуточный язык. В Первых реализациях языков C++, Objective-C, Go — код, написанный на этих языках, транслировался на язык C.
- Реализован для большинства аппаратных платформ и ОС.}

C – отличный язык для выражения общих идей в программировании способом, который удобен большинству программистов. Более того, многие принципы, используемые в C – например, argc и argv для параметров командной строки, будут отображаться во многих других языках, так что вы сможете общаться с людьми, даже если они не знают C, способом, который является общим для вас обоих. Эти сопоставления говорят о существенной «C-ориентации» мышления современных системных программистов.

C - фундаментальная основа квалификации программиста.

Библиотека libc

Библиотека libc была стандартизована в 1989 г. как часть стандарта ANSI C, который в 1990 г. был признан ISO (текущая актуальная версия стандарта, т. н. C18 – ISO/IEC 9899:2018 Information technology – Programming languages – C).

Библиотека состоит из нескольких десятков заголовочных файлов, подключаемых к программному проекту директивами. Эти файлы содержат описания одной или более функций, определения типов данных и макроопределения.

NB NB. В стандарте библиотека представлена как описание интерфейса прикладного программирования — API (Application Programming Interface) и для практического применения она должна быть реализована для конкретного компилятора и/или операционной системы.

Таким образом `libc` является неотъемлемой частью каждой реализации языка C.

Существует несколько десятков реализаций `libc`, в том числе такие широко применяемые как:

- GNU C Library — `glibc` — самая распространенная реализация, используемая в ОС Linux;
- Microsoft C Run-time Library, используемая в ОС Windows;
- `libSystem`, используемая в ОС iOS компании Apple;
- `Bionic`, разработанная компанией Google для ОС Android.

Библиотека `libc` оказала основополагающее влияние на реализации большинства современных системы программирования, среди которых C++, Python, Rust, GO и др. Например каждый заголовочный файл из стандартной библиотеки языка C включен в стандартную библиотеку языка C++ под именами, созданными путём отсечения в имени файла расширения `.h` и добавлением символа `'c'` в начале имени, например имени `'time.h'` соответствует `'ctime'`.

Единственное отличие между этими заголовочными файлами и традиционными заголовочными файлами стандартной библиотеки языка C заключается в том, что функции должны быть помещены в пространство имен `std::` (хотя некоторые компиляторы сами делают это).

Библиотека glibc

Библиотека glibc обновляется два раза в год. Ее функции, требующие работы в адресном пространстве ядра, используют системные вызовы. В руководстве к актуальной версии glibc функции разбиты на 36 групп различного назначения. Информацию о текущей версии glibc, установленной в конкретной версии ОС Linux можно получить командой:

```
/lib/libc.so.6
```

Ссылки на описание glibc:

<https://www.gnu.org/software/libc/manual/>

Общая характеристика glibc на 25.04.2021 {25.07.23} со стр.

<https://www.openhub.net/p/glibc>

Code Locations: git://sourceware.org/git/glibc.git

In a Nutshell, GNU C Library...

...

has had 37,147 {40307} commits made by 604 {720} contributors representing 1,302,579 {1419588} lines of code

...

is mostly written in C

with an average number of source code comments:

Across all C projects on Open Hub, 19% of all source code lines are comments. This holds true for GNU C Library as well.

...

has a well established, mature codebase maintained by a very large development team with increasing Y - O - Y commits

...

took an estimated 371 **{406}** years of effort (COCOMO model)

starting with its first commit in February, 1995

ending with its most recent commit 7 months ago

COCOMO - Constructive Cost Model

Данные на 25.04.2021 о распределении языков программирования

в коде со стр.:

https://www.openhub.net/p/glibc/analyses/latest/languages_summary

Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	971,675	238,172	19.7%	152,531	1,362,378	74.0%
Assembly	257,940	75,262	22.6%	40,667	373,869	20.3%
C++	17,017	6,662	28.1%	4,131	27,810	1.5%
Autoconf	15,117	153	1.0%	1,719	16,989	0.9%
Make	12,581	2,802	18.2%	2,759	18,142	1.0%
shell script	9,694	2,026	17.3%	1,177	12,897	0.7%
Python	8,863	3,897	30.5%	1,413	14,173	0.8%
TeX/LaTeX	7,163	3,695	34.0%	816	11,674	0.6%
AWK	1,898	373	16.4%	266	2,537	0.1%
Perl	631	143	18.5%	87	861	0.0%
Totals	1,302,579	333,185		205,566	1,841,330	

Освоив соглашения стандарта ABI о связях функций разработчик

получает доступ из языка ассемблера и из всех других языков и

систем программирования, поддерживающих ABI к примерно 1500

(количество зависит от версии и реализации) функций библиотеки

языка glibc, представляющей *золотой фонд программирования*.

Table of Contents

1 Introduction	19 Mathematics
2 Error Reporting	20 Arithmetic Functions
3 Virtual Memory Allocation And Paging	21 Date and Time
4 Character Handling	22 Resource Usage And Limitation
5 String and Array Utilities	23 Non-Local Exits
6 Character Set Handling	24 Signal Handling
7 Locales and Internationalization ns	25 The Basic Program/System Interface
8 Message Translation	26 Processes
9 Searching and Sorting	27 Inter-Process Communication
10 Pattern Matching	28 Job Control
11 Input/Output Overview	29 System Databases and Name Service
12 Input/Output on Streams	30 Users and Groups
13 Low-Level Input/Output	31 System Management
14 File System Interface	32 System Configuration Parameters
15 Pipes and FIFOs	33 Cryptographic Functions
16 Sockets	34 Debugging support
17 Low-Level Terminal Interface	35 Threads
18 Syslog	36 Internal probes

Полное содержание руководства по библиотеке glibc

Table of Contents

- **1 Introduction**
 - **1.1 Getting Started**
 - **1.2 Standards and Portability**
 - **1.2.1 ISO C**
 - **1.2.2 POSIX (The Portable Operating System Interface)**
 - **1.2.2.1 POSIX Safety Concepts**
 - **1.2.2.2 Unsafe Features**
 - **1.2.2.3 Conditionally Safe Features**
 - **1.2.2.4 Other Safety Remarks**
 - **1.2.3 Berkeley Unix**
 - **1.2.4 SVID (The System V Interface Description)**
 - **1.2.5 XPG (The X/Open Portability Guide)**
 - **1.3 Using the Library**

- 1.3.1 Header Files
- 1.3.2 Macro Definitions of Functions
- 1.3.3 Reserved Names
- 1.3.4 Feature Test Macros
- 1.4 Roadmap to the Manual
- 2 Error Reporting
 - 2.1 Checking for Errors
 - 2.2 Error Codes
 - 2.3 Error Messages
- 3 Virtual Memory Allocation And Paging
 - 3.1 Process Memory Concepts
 - 3.2 Allocating Storage For Program Data
 - 3.2.1 Memory Allocation in C Programs
 - 3.2.1.1 Dynamic Memory Allocation
 - 3.2.2 The GNU Allocator
 - 3.2.3 Unconstrained Allocation
 - 3.2.3.1 Basic Memory Allocation
 - 3.2.3.2 Examples of malloc
 - 3.2.3.3 Freeing Memory Allocated with malloc
 - 3.2.3.4 Changing the Size of a Block
 - 3.2.3.5 Allocating Cleared Space
 - 3.2.3.6 Allocating Aligned Memory Blocks
 - 3.2.3.7 Malloc Tunable Parameters
 - 3.2.3.8 Heap Consistency Checking
 - 3.2.3.9 Memory Allocation Hooks
 - 3.2.3.10 Statistics for Memory Allocation with malloc
 - 3.2.3.11 Summary of malloc-Related Functions
 - 3.2.4 Allocation Debugging
 - 3.2.4.1 How to install the tracing functionality
 - 3.2.4.2 Example program excerpts
 - 3.2.4.3 Some more or less clever ideas
 - 3.2.4.4 Interpreting the traces
 - 3.2.5 Replacing malloc
 - 3.2.6 Obstacks
 - 3.2.6.1 Creating Obstacks

- 3.2.6.2 Preparing for Using Obstacks
 - 3.2.6.3 Allocation in an Obstack
 - 3.2.6.4 Freeing Objects in an Obstack
 - 3.2.6.5 Obstack Functions and Macros
 - 3.2.6.6 Growing Objects
 - 3.2.6.7 Extra Fast Growing Objects
 - 3.2.6.8 Status of an Obstack
 - 3.2.6.9 Alignment of Data in Obstacks
 - 3.2.6.10 Obstack Chunks
 - 3.2.6.11 Summary of Obstack Functions
- 3.2.7 Automatic Storage with Variable Size
 - 3.2.7.1 alloca Example
 - 3.2.7.2 Advantages of alloca
 - 3.2.7.3 Disadvantages of alloca
 - 3.2.7.4 GNU C Variable-Size Arrays
- 3.3 Resizing the Data Segment
- 3.4 Memory Protection
 - 3.4.1 Memory Protection Keys
- 3.5 Locking Pages
 - 3.5.1 Why Lock Pages
 - 3.5.2 Locked Memory Details
 - 3.5.3 Functions To Lock And Unlock Pages
- 4 Character Handling
 - 4.1 Classification of Characters
 - 4.2 Case Conversion
 - 4.3 Character class determination for wide characters
 - 4.4 Notes on using the wide character classes
 - 4.5 Mapping of wide characters.
- 5 String and Array Utilities
 - 5.1 Representation of Strings
 - 5.2 String and Array Conventions
 - 5.3 String Length
 - 5.4 Copying Strings and Arrays
 - 5.5 Concatenating Strings
 - 5.6 Truncating Strings while Copying

- **5.7 String/Array Comparison**
- **5.8 Collation Functions**
- **5.9 Search Functions**
 - **5.9.1 Compatibility String Search Functions**
- **5.10 Finding Tokens in a String**
- **5.11 Erasing Sensitive Data**
- **5.12 Shuffling Bytes**
- **5.13 Obfuscating Data**
- **5.14 Encode Binary Data**
- **5.15 Argz and Envz Vectors**
 - **5.15.1 Argz Functions**
 - **5.15.2 Envz Functions**
- **6 Character Set Handling**
 - **6.1 Introduction to Extended Characters**
 - **6.2 Overview about Character Handling Functions**
 - **6.3 Restartable Multibyte Conversion Functions**
 - **6.3.1 Selecting the conversion and its properties**
 - **6.3.2 Representing the state of the conversion**
 - **6.3.3 Converting Single Characters**
 - **6.3.4 Converting Multibyte and Wide Character Strings**
 - **6.3.5 A Complete Multibyte Conversion Example**
 - **6.4 Non-reentrant Conversion Function**
 - **6.4.1 Non-reentrant Conversion of Single Characters**
 - **6.4.2 Non-reentrant Conversion of Strings**
 - **6.4.3 States in Non-reentrant Functions**
 - **6.5 Generic Charset Conversion**
 - **6.5.1 Generic Character Set Conversion Interface**
 - **6.5.2 A complete iconv example**
 - **6.5.3 Some Details about other iconv Implementations**
 - **6.5.4 The iconv Implementation in the GNU C Library**
 - **6.5.4.1 Format of gconv-modules files**
 - **6.5.4.2 Finding the conversion path in iconv**
 - **6.5.4.3 iconv module data structures**
 - **6.5.4.4 iconv module interfaces**
- **7 Locales and Internationalization**

- 7.1 What Effects a Locale Has
- 7.2 Choosing a Locale
- 7.3 Locale Categories
- 7.4 How Programs Set the Locale
- 7.5 Standard Locales
- 7.6 Locale Names
- 7.7 Accessing Locale Information
 - 7.7.1 localeconv: It is portable but ...
 - 7.7.1.1 Generic Numeric Formatting Parameters
 - 7.7.1.2 Printing the Currency Symbol
 - 7.7.1.3 Printing the Sign of a Monetary Amount
 - 7.7.2 Pinpoint Access to Locale Data
- 7.8 A dedicated function to format numbers
- 7.9 Yes-or-No Questions
- 8 Message Translation
 - 8.1 X/Open Message Catalog Handling
 - 8.1.1 The catgets function family
 - 8.1.2 Format of the message catalog files
 - 8.1.3 Generate Message Catalogs files
 - 8.1.4 How to use the catgets interface
 - 8.1.4.1 Not using symbolic names
 - 8.1.4.2 Using symbolic names
 - 8.1.4.3 How does to this allow to develop
 - 8.2 The Uniform approach to Message Translation
 - 8.2.1 The gettext family of functions
 - 8.2.1.1 What has to be done to translate a message?
 - 8.2.1.2 How to determine which catalog to be used
 - 8.2.1.3 Additional functions for more complicated situations
 - 8.2.1.4 How to specify the output character set gettext uses
 - 8.2.1.5 How to use gettext in GUI programs
 - 8.2.1.6 User influence on gettext
 - 8.2.2 Programs to handle message catalogs for gettext
- 9 Searching and Sorting

- **9.1 Defining the Comparison Function**
- **9.2 Array Search Function**
- **9.3 Array Sort Function**
- **9.4 Searching and Sorting Example**
- **9.5 The hsearch function.**
- **9.6 The tsearch function.**
- **10 Pattern Matching**
 - **10.1 Wildcard Matching**
 - **10.2 Globbing**
 - **10.2.1 Calling glob**
 - **10.2.2 Flags for Globbing**
 - **10.2.3 More Flags for Globbing**
 - **10.3 Regular Expression Matching**
 - **10.3.1 POSIX Regular Expression Compilation**
 - **10.3.2 Flags for POSIX Regular Expressions**
 - **10.3.3 Matching a Compiled POSIX Regular Expression**
 - **10.3.4 Match Results with Subexpressions**
 - **10.3.5 Complications in Subexpression Matching**
 - **10.3.6 POSIX Regexp Matching Cleanup**
 - **10.4 Shell-Style Word Expansion**
 - **10.4.1 The Stages of Word Expansion**
 - **10.4.2 Calling wordexp**
 - **10.4.3 Flags for Word Expansion**
 - **10.4.4 wordexp Example**
 - **10.4.5 Details of Tilde Expansion**
 - **10.4.6 Details of Variable Substitution**
- **11 Input/Output Overview**
 - **11.1 Input/Output Concepts**
 - **11.1.1 Streams and File Descriptors**
 - **11.1.2 File Position**
 - **11.2 File Names**
 - **11.2.1 Directories**
 - **11.2.2 File Name Resolution**
 - **11.2.3 File Name Errors**
 - **11.2.4 Portability of File Names**

- **12 Input/Output on Streams**
 - **12.1 Streams**
 - **12.2 Standard Streams**
 - **12.3 Opening Streams**
 - **12.4 Closing Streams**
 - **12.5 Streams and Threads**
 - **12.6 Streams in Internationalized Applications**
 - **12.7 Simple Output by Characters or Lines**
 - **12.8 Character Input**
 - **12.9 Line-Oriented Input**
 - **12.10 Unreading**
 - **12.10.1 What Unreading Means**
 - **12.10.2 Using `ungetc` To Do Unreading**
 - **12.11 Block Input/Output**
 - **12.12 Formatted Output**
 - **12.12.1 Formatted Output Basics**
 - **12.12.2 Output Conversion Syntax**
 - **12.12.3 Table of Output Conversions**
 - **12.12.4 Integer Conversions**
 - **12.12.5 Floating-Point Conversions**
 - **12.12.6 Other Output Conversions**
 - **12.12.7 Formatted Output Functions**
 - **12.12.8 Dynamically Allocating Formatted Output**
 - **12.12.9 Variable Arguments Output Functions**
 - **12.12.10 Parsing a Template String**
 - **12.12.11 Example of Parsing a Template String**
 - **12.13 Customizing `printf`**
 - **12.13.1 Registering New Conversions**
 - **12.13.2 Conversion Specifier Options**
 - **12.13.3 Defining the Output Handler**
 - **12.13.4 `printf` Extension Example**
 - **12.13.5 Predefined `printf` Handlers**
 - **12.14 Formatted Input**
 - **12.14.1 Formatted Input Basics**
 - **12.14.2 Input Conversion Syntax**

- 12.14.3 Table of Input Conversions
- 12.14.4 Numeric Input Conversions
- 12.14.5 String Input Conversions
- 12.14.6 Dynamically Allocating String Conversions
- 12.14.7 Other Input Conversions
- 12.14.8 Formatted Input Functions
- 12.14.9 Variable Arguments Input Functions
- 12.15 End-Of-File and Errors
- 12.16 Recovering from errors
- 12.17 Text and Binary Streams
- 12.18 File Positioning
- 12.19 Portable File-Position Functions
- 12.20 Stream Buffering
 - 12.20.1 Buffering Concepts
 - 12.20.2 Flushing Buffers
 - 12.20.3 Controlling Which Kind of Buffering
- 12.21 Other Kinds of Streams
 - 12.21.1 String Streams
 - 12.21.2 Programming Your Own Custom Streams
 - 12.21.2.1 Custom Streams and Cookies
 - 12.21.2.2 Custom Stream Hook Functions
- 12.22 Formatted Messages
 - 12.22.1 Printing Formatted Messages
 - 12.22.2 Adding Severity Classes
 - 12.22.3 How to use fmtmsg and addseverity
- 13 Low-Level Input/Output
 - 13.1 Opening and Closing Files
 - 13.2 Input and Output Primitives
 - 13.3 Setting the File Position of a Descriptor
 - 13.4 Descriptors and Streams
 - 13.5 Dangers of Mixing Streams and Descriptors
 - 13.5.1 Linked Channels
 - 13.5.2 Independent Channels
 - 13.5.3 Cleaning Streams
 - 13.6 Fast Scatter-Gather I/O

- 13.7 Copying data between two files
- 13.8 Memory-mapped I/O
- 13.9 Waiting for Input or Output
- 13.10 Synchronizing I/O operations
- 13.11 Perform I/O Operations in Parallel
 - 13.11.1 Asynchronous Read and Write Operations
 - 13.11.2 Getting the Status of AIO Operations
 - 13.11.3 Getting into a Consistent State
 - 13.11.4 Cancellation of AIO Operations
 - 13.11.5 How to optimize the AIO implementation
- 13.12 Control Operations on Files
- 13.13 Duplicating Descriptors
- 13.14 File Descriptor Flags
- 13.15 File Status Flags
 - 13.15.1 File Access Modes
 - 13.15.2 Open-time Flags
 - 13.15.3 I/O Operating Modes
 - 13.15.4 Getting and Setting File Status Flags
- 13.16 File Locks
- 13.17 Open File Description Locks
- 13.18 Open File Description Locks Example
- 13.19 Interrupt-Driven Input
- 13.20 Generic I/O Control operations
- 14 File System Interface
 - 14.1 Working Directory
 - 14.2 Accessing Directories
 - 14.2.1 Format of a Directory Entry
 - 14.2.2 Opening a Directory Stream
 - 14.2.3 Reading and Closing a Directory Stream
 - 14.2.4 Simple Program to List a Directory
 - 14.2.5 Random Access in a Directory Stream
 - 14.2.6 Scanning the Content of a Directory
 - 14.2.7 Simple Program to List a Directory, Mark II
 - 14.2.8 Low-level Directory Access
 - 14.3 Working with Directory Trees

- **14.4 Hard Links**
- **14.5 Symbolic Links**
- **14.6 Deleting Files**
- **14.7 Renaming Files**
- **14.8 Creating Directories**
- **14.9 File Attributes**
 - **14.9.1 The meaning of the File Attributes**
 - **14.9.2 Reading the Attributes of a File**
 - **14.9.3 Testing the Type of a File**
 - **14.9.4 File Owner**
 - **14.9.5 The Mode Bits for Access Permission**
 - **14.9.6 How Your Access to a File is Decided**
 - **14.9.7 Assigning File Permissions**
 - **14.9.8 Testing Permission to Access a File**
 - **14.9.9 File Times**
 - **14.9.10 File Size**
 - **14.9.11 Storage Allocation**
- **14.10 Making Special Files**
- **14.11 Temporary Files**
- **15 Pipes and FIFOs**
 - **15.1 Creating a Pipe**
 - **15.2 Pipe to a Subprocess**
 - **15.3 FIFO Special Files**
 - **15.4 Atomicity of Pipe I/O**
- **16 Sockets**
 - **16.1 Socket Concepts**
 - **16.2 Communication Styles**
 - **16.3 Socket Addresses**
 - **16.3.1 Address Formats**
 - **16.3.2 Setting the Address of a Socket**
 - **16.3.3 Reading the Address of a Socket**
 - **16.4 Interface Naming**
 - **16.5 The Local Namespace**
 - **16.5.1 Local Namespace Concepts**
 - **16.5.2 Details of Local Namespace**

- 16.5.3 Example of Local-Namespace Sockets
- 16.6 The Internet Namespace
 - 16.6.1 Internet Socket Address Formats
 - 16.6.2 Host Addresses
 - 16.6.2.1 Internet Host Addresses
 - 16.6.2.2 Host Address Data Type
 - 16.6.2.3 Host Address Functions
 - 16.6.2.4 Host Names
 - 16.6.3 Internet Ports
 - 16.6.4 The Services Database
 - 16.6.5 Byte Order Conversion
 - 16.6.6 Protocols Database
 - 16.6.7 Internet Socket Example
- 16.7 Other Namespaces
- 16.8 Opening and Closing Sockets
 - 16.8.1 Creating a Socket
 - 16.8.2 Closing a Socket
 - 16.8.3 Socket Pairs
- 16.9 Using Sockets with Connections
 - 16.9.1 Making a Connection
 - 16.9.2 Listening for Connections
 - 16.9.3 Accepting Connections
 - 16.9.4 Who is Connected to Me?
 - 16.9.5 Transferring Data
 - 16.9.5.1 Sending Data
 - 16.9.5.2 Receiving Data
 - 16.9.5.3 Socket Data Options
 - 16.9.6 Byte Stream Socket Example
 - 16.9.7 Byte Stream Connection Server Example
 - 16.9.8 Out-of-Band Data
- 16.10 Datagram Socket Operations
 - 16.10.1 Sending Datagrams
 - 16.10.2 Receiving Datagrams
 - 16.10.3 Datagram Socket Example
 - 16.10.4 Example of Reading Datagrams

- **16.11 The inetd Daemon**
 - **16.11.1 inetd Servers**
 - **16.11.2 Configuring inetd**
- **16.12 Socket Options**
 - **16.12.1 Socket Option Functions**
 - **16.12.2 Socket-Level Options**
- **16.13 Networks Database**
- **17 Low-Level Terminal Interface**
 - **17.1 Identifying Terminals**
 - **17.2 I/O Queues**
 - **17.3 Two Styles of Input: Canonical or Not**
 - **17.4 Terminal Modes**
 - **17.4.1 Terminal Mode Data Types**
 - **17.4.2 Terminal Mode Functions**
 - **17.4.3 Setting Terminal Modes Properly**
 - **17.4.4 Input Modes**
 - **17.4.5 Output Modes**
 - **17.4.6 Control Modes**
 - **17.4.7 Local Modes**
 - **17.4.8 Line Speed**
 - **17.4.9 Special Characters**
 - **17.4.9.1 Characters for Input Editing**
 - **17.4.9.2 Characters that Cause Signals**
 - **17.4.9.3 Special Characters for Flow Control**
 - **17.4.9.4 Other Special Characters**
 - **17.4.10 Noncanonical Input**
 - **17.5 BSD Terminal Modes**
 - **17.6 Line Control Functions**
 - **17.7 Noncanonical Mode Example**
 - **17.8 Reading Passphrases**
 - **17.9 Pseudo-Terminals**
 - **17.9.1 Allocating Pseudo-Terminals**
 - **17.9.2 Opening a Pseudo-Terminal Pair**
- **18 Syslog**
 - **18.1 Overview of Syslog**

- **18.2 Submitting Syslog Messages**
 - **18.2.1 openlog**
 - **18.2.2 syslog, vsyslog**
 - **18.2.3 closelog**
 - **18.2.4 setlogmask**
 - **18.2.5 Syslog Example**
- **19 Mathematics**
 - **19.1 Predefined Mathematical Constants**
 - **19.2 Trigonometric Functions**
 - **19.3 Inverse Trigonometric Functions**
 - **19.4 Exponentiation and Logarithms**
 - **19.5 Hyperbolic Functions**
 - **19.6 Special Functions**
 - **19.7 Known Maximum Errors in Math Functions**
 - **19.8 Pseudo-Random Numbers**
 - **19.8.1 ISO C Random Number Functions**
 - **19.8.2 BSD Random Number Functions**
 - **19.8.3 SVID Random Number Function**
 - **19.9 Is Fast Code or Small Code preferred?**
- **20 Arithmetic Functions**
 - **20.1 Integers**
 - **20.2 Integer Division**
 - **20.3 Floating Point Numbers**
 - **20.4 Floating-Point Number Classification Functions**
 - **20.5 Errors in Floating-Point Calculations**
 - **20.5.1 FP Exceptions**
 - **20.5.2 Infinity and NaN**
 - **20.5.3 Examining the FPU status word**
 - **20.5.4 Error Reporting by Mathematical Functions**
 - **20.6 Rounding Modes**
 - **20.7 Floating-Point Control Functions**
 - **20.8 Arithmetic Functions**
 - **20.8.1 Absolute Value**
 - **20.8.2 Normalization Functions**
 - **20.8.3 Rounding Functions**

- 20.8.4 Remainder Functions
- 20.8.5 Setting and modifying single bits of FP values
- 20.8.6 Floating-Point Comparison Functions
- 20.8.7 Miscellaneous FP arithmetic functions
- 20.9 Complex Numbers
- 20.10 Projections, Conjugates, and Decomposing of Complex Numbers
- 20.11 Parsing of Numbers
 - 20.11.1 Parsing of Integers
 - 20.11.2 Parsing of Floats
- 20.12 Printing of Floats
- 20.13 Old-fashioned System V number-to-string functions
- 21 Date and Time
 - 21.1 Time Basics
 - 21.2 Time Types
 - 21.3 Calculating Elapsed Time
 - 21.4 Processor And CPU Time
 - 21.4.1 CPU Time Inquiry
 - 21.4.2 Processor Time Inquiry
 - 21.5 Calendar Time
 - 21.5.1 Getting the Time
 - 21.5.2 Setting and Adjusting the Time
 - 21.5.3 Broken-down Time
 - 21.5.4 Formatting Calendar Time
 - 21.5.5 Convert textual time and date information back
 - 21.5.5.1 Interpret string according to given format
 - 21.5.5.2 A More User-friendly Way to Parse Times and Dates
 - 21.5.6 Specifying the Time Zone with TZ
 - 21.5.7 Functions and Variables for Time Zones
 - 21.5.8 Time Functions Example
 - 21.6 Setting an Alarm
 - 21.7 Sleeping
- 22 Resource Usage And Limitation
 - 22.1 Resource Usage

- 22.2 Limiting Resource Usage
- 22.3 Process CPU Priority And Scheduling
 - 22.3.1 Absolute Priority
 - 22.3.1.1 Using Absolute Priority
 - 22.3.2 Realtime Scheduling
 - 22.3.3 Basic Scheduling Functions
 - 22.3.4 Traditional Scheduling
 - 22.3.4.1 Introduction To Traditional Scheduling
 - 22.3.4.2 Functions For Traditional Scheduling
 - 22.3.5 Limiting execution to certain CPUs
- 22.4 Querying memory available resources
 - 22.4.1 Overview about traditional Unix memory handling
 - 22.4.2 How to get information about the memory subsystem?
- 22.5 Learn about the processors available
- 23 Non-Local Exits
 - 23.1 Introduction to Non-Local Exits
 - 23.2 Details of Non-Local Exits
 - 23.3 Non-Local Exits and Signals
 - 23.4 Complete Context Control
- 24 Signal Handling
 - 24.1 Basic Concepts of Signals
 - 24.1.1 Some Kinds of Signals
 - 24.1.2 Concepts of Signal Generation
 - 24.1.3 How Signals Are Delivered
 - 24.2 Standard Signals
 - 24.2.1 Program Error Signals
 - 24.2.2 Termination Signals
 - 24.2.3 Alarm Signals
 - 24.2.4 Asynchronous I/O Signals
 - 24.2.5 Job Control Signals
 - 24.2.6 Operation Error Signals
 - 24.2.7 Miscellaneous Signals
 - 24.2.8 Signal Messages
 - 24.3 Specifying Signal Actions
 - 24.3.1 Basic Signal Handling

- 24.3.2 Advanced Signal Handling
- 24.3.3 Interaction of signal and sigaction
- 24.3.4 sigaction Function Example
- 24.3.5 Flags for sigaction
- 24.3.6 Initial Signal Actions
- 24.4 Defining Signal Handlers
 - 24.4.1 Signal Handlers that Return
 - 24.4.2 Handlers That Terminate the Process
 - 24.4.3 Nonlocal Control Transfer in Handlers
 - 24.4.4 Signals Arriving While a Handler Runs
 - 24.4.5 Signals Close Together Merge into One
 - 24.4.6 Signal Handling and Nonreentrant Functions
 - 24.4.7 Atomic Data Access and Signal Handling
 - 24.4.7.1 Problems with Non-Atomic Access
 - 24.4.7.2 Atomic Types
 - 24.4.7.3 Atomic Usage Patterns
- 24.5 Primitives Interrupted by Signals
- 24.6 Generating Signals
 - 24.6.1 Signaling Yourself
 - 24.6.2 Signaling Another Process
 - 24.6.3 Permission for using kill
 - 24.6.4 Using kill for Communication
- 24.7 Blocking Signals
 - 24.7.1 Why Blocking Signals is Useful
 - 24.7.2 Signal Sets
 - 24.7.3 Process Signal Mask
 - 24.7.4 Blocking to Test for Delivery of a Signal
 - 24.7.5 Blocking Signals for a Handler
 - 24.7.6 Checking for Pending Signals
 - 24.7.7 Remembering a Signal to Act On Later
- 24.8 Waiting for a Signal
 - 24.8.1 Using pause
 - 24.8.2 Problems with pause
 - 24.8.3 Using sigsuspend
- 24.9 Using a Separate Signal Stack

- 24.10 BSD Signal Handling
- 25 The Basic Program/System Interface
 - 25.1 Program Arguments
 - 25.1.1 Program Argument Syntax Conventions
 - 25.1.2 Parsing Program Arguments
 - 25.2 Parsing program options using getopt
 - 25.2.1 Using the getopt function
 - 25.2.2 Example of Parsing Arguments with getopt
 - 25.2.3 Parsing Long Options with getopt_long
 - 25.2.4 Example of Parsing Long Options with getopt_long
 - 25.3 Parsing Program Options with Argp
 - 25.3.1 The argp_parse Function
 - 25.3.2 Argp Global Variables
 - 25.3.3 Specifying Argp Parsers
 - 25.3.4 Specifying Options in an Argp Parser
 - 25.3.4.1 Flags for Argp Options
 - 25.3.5 Argp Parser Functions
 - 25.3.5.1 Special Keys for Argp Parser Functions
 - 25.3.5.2 Argp Parsing State
 - 25.3.5.3 Functions For Use in Argp Parsers
 - 25.3.6 Combining Multiple Argp Parsers
 - 25.3.7 Flags for argp_parse
 - 25.3.8 Customizing Argp Help Output
 - 25.3.8.1 Special Keys for Argp Help Filter Functions
 - 25.3.9 The argp_help Function
 - 25.3.10 Flags for the argp_help Function
 - 25.3.11 Argp Examples
 - 25.3.11.1 A Minimal Program Using Argp
 - 25.3.11.2 A Program Using Argp with Only Default Options
 - 25.3.11.3 A Program Using Argp with User Options
 - 25.3.11.4 A Program Using Multiple Combined Argp Parsers
 - 25.3.12 Argp User Customization
 - 25.3.12.1 Parsing of Suboptions

- 25.3.13 Parsing of Suboptions Example
- 25.4 Environment Variables
 - 25.4.1 Environment Access
 - 25.4.2 Standard Environment Variables
- 25.5 Auxiliary Vector
 - 25.5.1 Definition of getauxval
- 25.6 System Calls
- 25.7 Program Termination
 - 25.7.1 Normal Termination
 - 25.7.2 Exit Status
 - 25.7.3 Cleanups on Exit
 - 25.7.4 Aborting a Program
 - 25.7.5 Termination Internals
- 26 Processes
 - 26.1 Running a Command
 - 26.2 Process Creation Concepts
 - 26.3 Process Identification
 - 26.4 Creating a Process
 - 26.5 Executing a File
 - 26.6 Process Completion
 - 26.7 Process Completion Status
 - 26.8 BSD Process Wait Function
 - 26.9 Process Creation Example
- 27 Inter-Process Communication
 - 27.1 Semaphores
 - 27.1.1 System V Semaphores
 - 27.1.2 POSIX Semaphores
- 28 Job Control
 - 28.1 Concepts of Job Control
 - 28.2 Controlling Terminal of a Process
 - 28.3 Access to the Controlling Terminal
 - 28.4 Orphaned Process Groups
 - 28.5 Implementing a Job Control Shell
 - 28.5.1 Data Structures for the Shell
 - 28.5.2 Initializing the Shell

- 28.5.3 Launching Jobs
- 28.5.4 Foreground and Background
- 28.5.5 Stopped and Terminated Jobs
- 28.5.6 Continuing Stopped Jobs
- 28.5.7 The Missing Pieces
- 28.6 Functions for Job Control
 - 28.6.1 Identifying the Controlling Terminal
 - 28.6.2 Process Group Functions
 - 28.6.3 Functions for Controlling Terminal Access
- 29 System Databases and Name Service Switch
 - 29.1 NSS Basics
 - 29.2 The NSS Configuration File
 - 29.2.1 Services in the NSS configuration File
 - 29.2.2 Actions in the NSS configuration
 - 29.2.3 Notes on the NSS Configuration File
 - 29.3 NSS Module Internals
 - 29.3.1 The Naming Scheme of the NSS Modules
 - 29.3.2 The Interface of the Function in NSS Modules
 - 29.4 Extending NSS
 - 29.4.1 Adding another Service to NSS
 - 29.4.2 Internals of the NSS Module Functions
- 30 Users and Groups
 - 30.1 User and Group IDs
 - 30.2 The Persona of a Process
 - 30.3 Why Change the Persona of a Process?
 - 30.4 How an Application Can Change Persona
 - 30.5 Reading the Persona of a Process
 - 30.6 Setting the User ID
 - 30.7 Setting the Group IDs
 - 30.8 Enabling and Disabling Setuid Access
 - 30.9 Setuid Program Example
 - 30.10 Tips for Writing Setuid Programs
 - 30.11 Identifying Who Logged In
 - 30.12 The User Accounting Database
 - 30.12.1 Manipulating the User Accounting Database

- 30.12.2 XPG User Accounting Database Functions
 - 30.12.3 Logging In and Out
- 30.13 User Database
 - 30.13.1 The Data Structure that Describes a User
 - 30.13.2 Looking Up One User
 - 30.13.3 Scanning the List of All Users
 - 30.13.4 Writing a User Entry
- 30.14 Group Database
 - 30.14.1 The Data Structure for a Group
 - 30.14.2 Looking Up One Group
 - 30.14.3 Scanning the List of All Groups
- 30.15 User and Group Database Example
- 30.16 Netgroup Database
 - 30.16.1 Netgroup Data
 - 30.16.2 Looking up one Netgroup
 - 30.16.3 Testing for Netgroup Membership
- 31 System Management
 - 31.1 Host Identification
 - 31.2 Platform Type Identification
 - 31.3 Controlling and Querying Mounts
 - 31.3.1 Mount Information
 - 31.3.1.1 The fstab file
 - 31.3.1.2 The mtab file
 - 31.3.1.3 Other (Non-libc) Sources of Mount Information
 - 31.3.2 Mount, Unmount, Remount
- 32 System Configuration Parameters
 - 32.1 General Capacity Limits
 - 32.2 Overall System Options
 - 32.3 Which Version of POSIX is Supported
 - 32.4 Using sysconf
 - 32.4.1 Definition of sysconf
 - 32.4.2 Constants for sysconf Parameters
 - 32.4.3 Examples of sysconf
 - 32.5 Minimum Values for General Capacity Limits
 - 32.6 Limits on File System Capacity

- **32.7 Optional Features in File Support**
- **32.8 Minimum Values for File System Limits**
- **32.9 Using pathconf**
- **32.10 Utility Program Capacity Limits**
- **32.11 Minimum Values for Utility Limits**
- **32.12 String-Valued Parameters**
- **33 Cryptographic Functions**
 - **33.1 Passphrase Storage**
 - **33.2 Generating Unpredictable Bytes**
- **34 Debugging support**
 - **34.1 Backtraces**
- **35 Threads**
 - **35.1 ISO C Threads**
 - **35.1.1 Return Values**
 - **35.1.2 Creation and Control**
 - **35.1.3 Call Once**
 - **35.1.4 Mutexes**
 - **35.1.5 Condition Variables**
 - **35.1.6 Thread-local Storage**
 - **35.2 POSIX Threads**
 - **35.2.1 Thread-specific Data**
 - **35.2.2 Non-POSIX Extensions**
 - **35.2.2.1 Setting Process-wide defaults for thread attributes**
 - **35.2.2.2 Controlling the Initial Signal Mask of a New Thread**
 - **35.2.2.3 Functions for Waiting According to a Specific Clock**
 - **35.2.2.4 Detecting Single-Threaded Execution**
- **36 Internal probes**
 - **36.1 Memory Allocation Probes**
 - **36.2 Mathematical Function Probes**
 - **36.3 Non-local Goto Probes**
- **37 Tunables**
 - **37.1 Tunable names**

- **37.2 Memory Allocation Tunables**
- **37.3 Dynamic Linking Tunables**
- **37.4 Elision Tunables**
- **37.5 POSIX Thread Tunables**
- **37.6 Hardware Capability Tunables**
- **Appendix A C Language Facilities in the Library**
 - **A.1 Explicitly Checking Internal Consistency**
 - **A.2 Variadic Functions**
 - **A.2.1 Why Variadic Functions are Used**
 - **A.2.2 How Variadic Functions are Defined and Used**
 - **A.2.2.1 Syntax for Variable Arguments**
 - **A.2.2.2 Receiving the Argument Values**
 - **A.2.2.3 How Many Arguments Were Supplied**
 - **A.2.2.4 Calling Variadic Functions**
 - **A.2.2.5 Argument Access Macros**
 - **A.2.3 Example of a Variadic Function**
 - **A.3 Null Pointer Constant**
 - **A.4 Important Data Types**
 - **A.5 Data Type Measurements**
 - **A.5.1 Width of an Integer Type**
 - **A.5.2 Range of an Integer Type**
 - **A.5.3 Floating Type Macros**
 - **A.5.3.1 Floating Point Representation Concepts**
 - **A.5.3.2 Floating Point Parameters**
 - **A.5.3.3 IEEE Floating Point**
 - **A.5.4 Structure Field Offset Measurement**
- **Appendix B Summary of Library Facilities**
- **Appendix C Installing the GNU C Library**
 - **C.1 Configuring and compiling the GNU C Library**
 - **C.2 Installing the C Library**
 - **C.3 Recommended Tools for Compilation**
 - **C.4 Specific advice for GNU/Linux systems**
 - **C.5 Reporting Bugs**
- **Appendix D Library Maintenance**
 - **D.1 Adding New Functions**

- D.1.1 Platform-specific types, macros and functions
- D.2 Symbol handling in the GNU C Library
 - D.2.1 64-bit time symbol handling in the GNU C Library
- D.3 Porting the GNU C Library
 - D.3.1 Layout of the sysdeps Directory Hierarchy
 - D.3.2 Porting the GNU C Library to Unix Systems
- Appendix E Platform-specific facilities
 - E.1 PowerPC-specific Facilities
 - E.2 RISC-V-specific Facilities
- Appendix F Contributors to the GNU C Library
- Appendix G Free Software Needs Free Documentation
- Appendix H GNU Lesser General Public License
- Appendix I GNU Free Documentation License
- Concept Index
- Type Index
- Function and Macro Index
- Variable and Constant Macro Index
- Program and File Index

Текст подключаемый препроцессором C к файлу `call-as.c`

Для того чтобы продемонстрировать масштаб `glibc` и объем проделанной работы рассмотрим какой текст подключает препроцессор C к файлу `call-as.c` из предыдущей лекции.

```
#include <stdio.h>
/* прототип внешней функции.
   Параметр 1 - количество символов, кот. надо обработать
   Параметр 2 - адрес первого элемента массива
*/
extern void *Read_Sym(int, char*);
/* массив результатов*/
```

```

int Numbers[10];
int main()
{
    char Symbols[14] = "91A23B456C789"; /* исходный массив */
    int i;
    printf ("%s\n", Symbols);
    Read_Sym(8, Symbols);                /* вызов as функции */
    /* печать результата */
    for (i = 0; i < 8; i++)
        printf("%d\t", Numbers[i]);
    return 0;
}

```

Код, формируемый препроцессором, можно получить командой:

```
gcc -m32 -gstabs+ -E -P call-as.c > call-as.1pp
```

Приведенный ниже файл call-as.1pp объемом 306 строк содержит код, полученный препроцессором для фазы компиляции из файла call-as.c объемом 46 строк.

```

typedef unsigned int size_t;
typedef unsigned char __u_char;
typedef unsigned short int __u_short;
typedef unsigned int __u_int;
typedef unsigned long int __u_long;
typedef signed char __int8_t;
typedef unsigned char __uint8_t;
typedef signed short int __int16_t;
typedef unsigned short int __uint16_t;
typedef signed int __int32_t;
typedef unsigned int __uint32_t;

```

```
__extension__ typedef signed long long int __int64_t;
__extension__ typedef unsigned long long int __uint64_t;
__extension__ typedef long long int __quad_t;
__extension__ typedef unsigned long long int __u_quad_t;
__extension__ typedef long long int __intmax_t;
__extension__ typedef unsigned long long int __uintmax_t;
__extension__ typedef __u_quad_t __dev_t;
__extension__ typedef unsigned int __uid_t;
__extension__ typedef unsigned int __gid_t;
__extension__ typedef unsigned long int __ino_t;
__extension__ typedef __u_quad_t __ino64_t;
__extension__ typedef unsigned int __mode_t;
__extension__ typedef unsigned int __nlink_t;
__extension__ typedef long int __off_t;
__extension__ typedef __quad_t __off64_t;
__extension__ typedef int __pid_t;
__extension__ typedef struct { int __val[2]; } __fsid_t;
__extension__ typedef long int __clock_t;
__extension__ typedef unsigned long int __rlim_t;
__extension__ typedef __u_quad_t __rlim64_t;
__extension__ typedef unsigned int __id_t;
__extension__ typedef long int __time_t;
__extension__ typedef unsigned int __useconds_t;
__extension__ typedef long int __suseconds_t;
__extension__ typedef int __daddr_t;
__extension__ typedef int __key_t;
__extension__ typedef int __clockid_t;
__extension__ typedef void * __timer_t;
__extension__ typedef long int __blksize_t;
__extension__ typedef long int __blkcnt_t;
__extension__ typedef __quad_t __blkcnt64_t;
__extension__ typedef unsigned long int __fsblkcnt_t;
__extension__ typedef __u_quad_t __fsblkcnt64_t;
```

```

__extension__ typedef unsigned long int __fsfilcnt_t;
__extension__ typedef __u_quad_t __fsfilcnt64_t;
__extension__ typedef int __fsword_t;
__extension__ typedef int __ssize_t;
__extension__ typedef long int __syscall_slong_t;
__extension__ typedef unsigned long int __syscall_ulong_t;
typedef __off64_t __loff_t;
typedef __quad_t *__qaddr_t;
typedef char *__caddr_t;
__extension__ typedef int __intptr_t;
__extension__ typedef unsigned int __socklen_t;
typedef int __sig_atomic_t;
struct _IO_FILE;
typedef struct _IO_FILE __FILE;
struct _IO_FILE;
typedef struct _IO_FILE FILE;
typedef struct
{
    int __count;
    union
    {
        unsigned int __wch;
        char __wchb[4];
    } __value;
} __mbstate_t;
typedef struct
{
    __off_t __pos;
    __mbstate_t __state;
} _G_fpos_t;
typedef struct
{
    __off64_t __pos;

```



```
__mbstate_t __state;
} _G_fpos64_t;
typedef __builtin_va_list __gnuc_va_list;
struct _IO_jump_t; struct _IO_FILE;
typedef void _IO_lock_t;
struct _IO_marker {
    struct _IO_marker *_next;
    struct _IO_FILE *_sbuf;
    int _pos;
};
enum __codecvt_result
{
    __codecvt_ok,
    __codecvt_partial,
    __codecvt_error,
    __codecvt_noconv
};
struct _IO_FILE {
    int _flags;
    char* _IO_read_ptr;
    char* _IO_read_end;
    char* _IO_read_base;
    char* _IO_write_base;
    char* _IO_write_ptr;
    char* _IO_write_end;
    char* _IO_buf_base;
    char* _IO_buf_end;
    char *_IO_save_base;
    char *_IO_backup_base;
    char *_IO_save_end;
    struct _IO_marker *_markers;
    struct _IO_FILE *_chain;
    int _fileno;
```

```

int _flags2;
__off_t _old_offset;
unsigned short _cur_column;
signed char _vtable_offset;
char _shortbuf[1];
_IO_lock_t *_lock;
__off64_t _offset;
void *__pad1;
void *__pad2;
void *__pad3;
void *__pad4;
size_t __pad5;
int _mode;
char _unused2[15 * sizeof (int) - 4 * sizeof (void *) -
sizeof (size_t)];
};
typedef struct _IO_FILE _IO_FILE;
struct _IO_FILE_plus;
extern struct _IO_FILE_plus _IO_2_1_stdin_;
extern struct _IO_FILE_plus _IO_2_1_stdout_;
extern struct _IO_FILE_plus _IO_2_1_stderr_;
typedef __ssize_t __io_read_fn (void *__cookie, char
*__buf, size_t __nbytes);
typedef __ssize_t __io_write_fn (void *__cookie, const
char *__buf,
    size_t __n);
typedef int __io_seek_fn (void *__cookie, __off64_t
*__pos, int __w);
typedef int __io_close_fn (void *__cookie);
extern int __underflow (_IO_FILE *);
extern int __uflow (_IO_FILE *);
extern int __overflow (_IO_FILE *, int);
extern int _IO_getc (_IO_FILE *__fp);

```

```
extern int _IO_putc (int __c, _IO_FILE *__fp);
extern int _IO_feof (_IO_FILE *__fp) __attribute__
((__nothrow__ , __leaf__));
extern int _IO_ferror (_IO_FILE *__fp) __attribute__
((__nothrow__ , __leaf__));
extern int _IO_peekc_locked (_IO_FILE *__fp);
extern void _IO_flockfile (_IO_FILE *) __attribute__
((__nothrow__ , __leaf__));
extern void _IO_funlockfile (_IO_FILE *) __attribute__
((__nothrow__ , __leaf__));
extern int _IO_ftrylockfile (_IO_FILE *) __attribute__
((__nothrow__ , __leaf__));
```

```
extern int _IO_vfscanf (_IO_FILE * __restrict, const char
* __restrict, __gnuc_va_list, int *__restrict);
extern int _IO_vfprintf (_IO_FILE *__restrict, const char
* __restrict, __gnuc_va_list);
```

```
extern __ssize_t _IO_padn (_IO_FILE *, int, __ssize_t);
extern size_t _IO_sgetn (_IO_FILE *, void *, size_t);
extern __off64_t _IO_seekoff (_IO_FILE *, __off64_t, int,
int);
extern __off64_t _IO_seekpos (_IO_FILE *, __off64_t, int);
extern void _IO_free_backup_area (_IO_FILE *)
__attribute__ ((__nothrow__ , __leaf__));
typedef __gnuc_va_list va_list;
typedef __off_t off_t;
typedef __ssize_t ssize_t;
typedef _G_fpos_t fpos_t;
extern struct _IO_FILE *stdin;
extern struct _IO_FILE *stdout;
extern struct _IO_FILE *stderr;
```

```

extern int remove (const char *__filename) __attribute__
((__nothrow__ , __leaf__));
extern int rename (const char *__old, const char *__new)
__attribute__ ((__nothrow__ , __leaf__));
extern int renameat (int __oldfd, const char *__old, int
__newfd,
    const char *__new) __attribute__ ((__nothrow__ ,
__leaf__));
extern FILE *tmpfile (void) ;
extern char *tmpnam (char *__s) __attribute__
((__nothrow__ , __leaf__)) ;
extern char *tmpnam_r (char *__s) __attribute__
((__nothrow__ , __leaf__)) ;
extern char *tempnam (const char *__dir, const char
*__pfx)
    __attribute__ ((__nothrow__ , __leaf__))
__attribute__ ((__malloc__)) ;
extern int fclose (FILE *__stream);
extern int fflush (FILE *__stream);
extern int fflush_unlocked (FILE *__stream);
extern FILE *fopen (const char *__restrict __filename,
    const char *__restrict __modes) ;
extern FILE *freopen (const char *__restrict __filename,
    const char *__restrict __modes,
    FILE *__restrict __stream) ;
extern FILE *fdopen (int __fd, const char *__modes)
__attribute__ ((__nothrow__ , __leaf__)) ;
extern FILE *fmemopen (void *__s, size_t __len, const char
*__modes)
    __attribute__ ((__nothrow__ , __leaf__)) ;
extern FILE *open_memstream (char **__bufloc, size_t
*__sizeloc) __attribute__ ((__nothrow__ , __leaf__)) ;

```

```
extern void setbuf (FILE *__restrict __stream, char
*__restrict __buf) __attribute__ ((__nothrow__ , __leaf__));
extern int setvbuf (FILE *__restrict __stream, char
*__restrict __buf,
    int __modes, size_t __n) __attribute__ ((__nothrow__
, __leaf__));
extern void setbuffer (FILE *__restrict __stream, char
*__restrict __buf,
    size_t __size) __attribute__ ((__nothrow__ ,
__leaf__));
extern void setlinebuf (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));
```

```
extern int fprintf (FILE *__restrict __stream,
    const char *__restrict __format, ...);
extern int printf (const char *__restrict __format, ...);
extern int sprintf (char *__restrict __s,
    const char *__restrict __format, ...) __attribute__
((__nothrow__));
extern int vfprintf (FILE *__restrict __s, const char
*__restrict __format,
    __gnuc_va_list __arg);
extern int vprintf (const char *__restrict __format,
__gnuc_va_list __arg);
extern int vsprintf (char *__restrict __s, const char
*__restrict __format,
    __gnuc_va_list __arg) __attribute__
((__nothrow__));
extern int snprintf (char *__restrict __s, size_t
__maxlen,
    const char *__restrict __format, ...)
```

```

        __attribute__((__nothrow__)) __attribute__((
((__format__ (__printf__, 3, 4))));
extern int vsnprintf (char *__restrict __s, size_t
__maxlen,
        const char *__restrict __format, __gnuc_va_list
__arg)
        __attribute__((__nothrow__)) __attribute__((
((__format__ (__printf__, 3, 0))));
extern int vdprintf (int __fd, const char *__restrict
__fmt,
        __gnuc_va_list __arg)
        __attribute__((__format__ (__printf__, 2, 0)));
extern int dprintf (int __fd, const char *__restrict
__fmt, ...)
        __attribute__((__format__ (__printf__, 2, 3)));

```

```

extern int fscanf (FILE *__restrict __stream,
        const char *__restrict __format, ...) ;
extern int scanf (const char *__restrict __format, ...) ;
extern int sscanf (const char *__restrict __s,
        const char *__restrict __format, ...) __attribute__((
((__nothrow__ , __leaf__)));
extern int fscanf (FILE *__restrict __stream, const char
*__restrict __format, ...) __asm__ ("\" __isoc99_fscanf") ;
extern int scanf (const char *__restrict __format, ...)
__asm__ ("\" __isoc99_scanf") ;
extern int sscanf (const char *__restrict __s, const char
*__restrict __format, ...) __asm__ ("\" __isoc99_sscanf")
__attribute__((__nothrow__ , __leaf__));

```

```

extern int vfscanf (FILE *__restrict __s, const char
*__restrict __format,
    __gnuc_va_list __arg)
    __attribute__ ((__format__ (__scanf__, 2, 0))) ;
extern int vscanf (const char *__restrict __format,
__gnuc_va_list __arg)
    __attribute__ ((__format__ (__scanf__, 1, 0))) ;
extern int vsscanf (const char *__restrict __s,
    const char *__restrict __format, __gnuc_va_list
__arg)
    __attribute__ ((__nothrow__ , __leaf__))
__attribute__ ((__format__ (__scanf__, 2, 0)));
extern int vfscanf (FILE *__restrict __s, const char
*__restrict __format, __gnuc_va_list __arg) __asm__ ("
"__isoc99_vfscanf")
    __attribute__ ((__format__ (__scanf__, 2, 0))) ;
extern int vscanf (const char *__restrict __format,
__gnuc_va_list __arg) __asm__ (" " "__isoc99_vscanf")
    __attribute__ ((__format__ (__scanf__, 1, 0))) ;
extern int vsscanf (const char *__restrict __s, const char
*__restrict __format, __gnuc_va_list __arg) __asm__ ("
"__isoc99_vsscanf") __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__format__ (__scanf__, 2, 0)));

```

```

extern int fgetc (FILE *__stream);
extern int getc (FILE *__stream);
extern int getchar (void);
extern int getc_unlocked (FILE *__stream);
extern int getchar_unlocked (void);
extern int fgetc_unlocked (FILE *__stream);
extern int fputc (int __c, FILE *__stream);
extern int putc (int __c, FILE *__stream);
extern int putchar (int __c);

```

```

extern int fputc_unlocked (int __c, FILE *__stream);
extern int putc_unlocked (int __c, FILE *__stream);
extern int putchar_unlocked (int __c);
extern int getw (FILE *__stream);
extern int putw (int __w, FILE *__stream);
extern char *fgets (char *__restrict __s, int __n, FILE
*__restrict __stream)
    ;
extern __ssize_t __getdelim (char **__restrict __lineptr,
    size_t *__restrict __n, int __delimiter,
    FILE *__restrict __stream) ;
extern __ssize_t getdelim (char **__restrict __lineptr,
    size_t *__restrict __n, int __delimiter,
    FILE *__restrict __stream) ;
extern __ssize_t getline (char **__restrict __lineptr,
    size_t *__restrict __n,
    FILE *__restrict __stream) ;
extern int fputs (const char *__restrict __s, FILE
*__restrict __stream);
extern int puts (const char *__s);
extern int ungetc (int __c, FILE *__stream);
extern size_t fread (void *__restrict __ptr, size_t
__size,
    size_t __n, FILE *__restrict __stream) ;
extern size_t fwrite (const void *__restrict __ptr, size_t
__size,
    size_t __n, FILE *__restrict __s);
extern size_t fread_unlocked (void *__restrict __ptr,
size_t __size,
    size_t __n, FILE *__restrict __stream) ;
extern size_t fwrite_unlocked (const void *__restrict
__ptr, size_t __size,
    size_t __n, FILE *__restrict __stream);

```



```
extern int fseek (FILE *__stream, long int __off, int
__whence);
extern long int ftell (FILE *__stream) ;
extern void rewind (FILE *__stream);
extern int fseeko (FILE *__stream, __off_t __off, int
__whence);
extern __off_t ftello (FILE *__stream) ;
extern int fgetpos (FILE *__restrict __stream, fpos_t
*__restrict __pos);
extern int fsetpos (FILE *__stream, const fpos_t *__pos);
extern void clearerr (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));
extern int feof (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern int ferror (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern void clearerr_unlocked (FILE *__stream)
__attribute__ ((__nothrow__ , __leaf__));
extern int feof_unlocked (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern int ferror_unlocked (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern void perror (const char *__s);
extern int sys_nerr;
extern const char *const sys_errlist[];
extern int fileno (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern int fileno_unlocked (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern FILE *popen (const char *__command, const char
*__modes) ;
extern int pclose (FILE *__stream);
```

```
extern char *ctermid (char *__s) __attribute__((__nothrow__ , __leaf__));
extern void flockfile (FILE *__stream) __attribute__((__nothrow__ , __leaf__));
extern int ftrylockfile (FILE *__stream) __attribute__((__nothrow__ , __leaf__)) ;
extern void funlockfile (FILE *__stream) __attribute__((__nothrow__ , __leaf__));

extern void *Read_Sym(int, char*);
int Numbers[10];
int main()
{
    char Symbols[14] = "91A23B456C789";
    int i;
    printf ("%s\n", Symbols);
    Read_Sym(8, Symbols);
    for (i = 0; i < 8; i++)
        printf("%d\t", Numbers[i]);
    return 0;
}
```

«Впечатляющий» список разнообразного ПО, разработанного на C с использованием `glibc` можно найти по ссылке:

<https://github.com/kozross/awesome-c>