ansi c（c90）

语法元素简明手册

目录

[序言：mcu c 和 ansi c](#前言) ...... 1

[关键字](#关键字) ..................... 1

[运算符](#运算符) ..................... 2

[分类](#分类)

[优先级](#优先级)

[基本数据类型](#数据类型) ............... 2

[基本数据类型](#基本数据类型)

[数据类型限定词](#限定词)

[基本数据类型转换](#基本数据类型转换)

[结构型数据类型](#结构型数据类型) ............ 3

[数组](#数组)

[指针](#指针)

[结构](#结构)

[联合](#联合)

[枚举](#枚举)

[流程控制语句](#流程控制语句) ............... 4

[复合语句](#复合语句)

[分枝语句](#分枝语句)

[循环语句](#循环语句)

[转向语句](#转向语句)

[函数](#函数) ....................... 5

[头文件](#头文件) ..................... 5

[附录 预处理命令和预定义宏](#预处理命令).. 5

[库函数](#库函数) ..................... 6

[字符处理函数](#字符处理函数)

[地区化处理函数](#地区化处理函数)

[数学函数](#数学函数)

[全局跳转函数](#全局跳转函数)

[信号处理函数](#信号处理函数)

[可变参数处理宏](#可变参数处理宏)

[输入输出函数](#输入输出函数)

[实用工具函数](#实用工具函数)

[字符串处理函数](#字符串处理函数)

[日期和时间函数](#日期和时间函数函数)

[c 51 和 picc 的特点](#c51和picc的特点) .......... 10

[C 语言发展简史](#c语言发展简史) ............... 11

序言：mcu c 和 ansi c

美国国家标准 ansi c, 是目前通行的 c 语言国际标准，现在流行的 mcu c 无一不遵循 ansi c，其基本语法元素，包括：关键字，运算符，基本数据类型，结构型数据类型，流程控制语句，函数，头文件，库函数等，都是与 ansi c 一致的，这是 mcu c 的共性。

每一种 mcu c 相对于 ansi c 都有所扩展和变通，形成了这种 mcu c 的特点。这些特点首先与具体的 mcu 的组织结构，指令系统，存储器的地址编排方式密切相关。而在表现形式上，应该主要出于软件设计者的偏好与灵感。

mcu c 的特点各各迥异，主要包括 3 个方面的扩展：① 中断函数标识方法。② 外围模块寄存器读写方法。③ 嵌入汇编语言语句和调用汇编语言函数。1 个方面的变通：与 mcu 结构相关的头文件，“标准头文件”的数量和内容，“标准库函数”的数量。除此之外，可能还会有一些个别的零散的特点，应该在使用中逐个地加以识别。

综合以上分析，遵循 ansi c 的 mcu c 的语法体系，应由两部分组成：一部分是作为共性的 ansi c 语法元素，一部分是这种 mcu c 特有的语法元素。因为我们对 ansi c 语法元素已经熟悉，所以，在使用一种新的 mcu c 的时候，只要注意了解它的语法特点，就可以收到事半功倍的效果了。

使用任何一种 mcu c ，都应对相应的 mcu 的组织结构，指令系统，外围模块等，有基本的了解。在这个基础上，研究一下开发环境附带的样例工程，或其它样例工程，包括这些样例工程中包含的启动代码文件，与单片机对应的头文件，用户源文件等。也可以参阅开发环境帮助目录中的“汇编语言参考”，“ c 语言参考”等文档。

ansi c 的代号为 ANSX3.159-1989，通常称为c89。国际标准化组织接受了这个标准，代号为ISO/IEC 9899:1990，通常称为c90。中国国家标准 GB/T 15272-94 程序设计语言 C ，等同采用 c90。



关键字

auto 声明自动变量

break 跳出当前循环

case 开关语句分支，与 switch 连用

char 声明字符型变量或函数

const 声明常量（或称只读变量）

continue 结束当前循环，开始下一轮循环

default 开关语句中的“其他”分支

do do 循环语句

double 声明双精度变量或函数

else if语句的否定分支

enum 定义枚举类型

extern 声明变量或函数在其他文件中声明

float 声明浮点型变量或函数

for for 循环语句

goto 无条件跳转语句

if 条件语句

int 声明整型变量或函数

long 声明长整型变量或函数

register 声明寄存器变量

return 子程序返回语句（带参数，或不带参数）

short 声明短整型变量或函数

signed 声明有符号类型变量或函数

sizeof 计算数据类型长度

static 声明静态变量

struct 定义结构类型

switch 开关语句

typedef 给数据类型取别名

union 定义联合类型

unsigned 声明无符号类型变量或函数

void 函数无返回值或无参数时，声明为void（空）

volatile 说明变量在程序执行中可被隐含地改变

while while 循环语句

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

volatile用法举例和说明：

volatile int i = 10;

int j = i;

...

int k = i;

volatile 告诉编译器，i在其它函数中，例如在中断服务函数中，是随时可能发生变化的，每次使用它的时候必须从 i 的地址中读取，因而编译器生成的可执行码会重新从 i 的地址读取数据放在k中，而优化做法是，由于编译器发现两次从 i 读数据的代码之间的代码没有对 i进行过操作，它会自动把上次读的数据放在 k 中，而不是重新从 i 里面读，这样，如果 i 是一个寄存器或者一个端口，或者其它全局变量，数据就容易出错，使用volatile，就可以保证对特殊地址的稳定访问，避免出错。

运算符

分类

算术运算 + - \* / %(取模) ++ --

逻辑运算 ! && ||

位逻辑运算 << >> ~ & | ^(异或)

指针运算 -> . \* &

关系运算 < <= > >= == !=

赋值运算 = += -= \*= /= %= &= ^= |= <<= >>=

其它运算 ()(强制转换) [](下标运算) +(正) -(负) sizeof ?:(3目运算) ,(逗号运算)

优先级 运算符 结合性

最高 () [] -> . 自左向右

! ~ ++ -- + - \* & sizeof(单目运算) 自右向左

( ~ 按位取反 + 正号 - 负号 \* 定义指针或取内容 & 取地址)

\* / % 自左向右

+ - (加减运算) 自左向右

<< >> 自左向右

< <= > >= 自左向右

== != 自左向右

& 自左向右

^ 自左向右

| 自左向右

&& 自左向右

|| 自左向右

?:(三目运算) 自右向左

= += -= \*= /= %= &= ^= |= <<= >>=(赋值运算) 自右向左

最低 , 自左向右

基本数据类型

基本数据类型

char 声明字符型变量或函数

int 声明整型变量或函数

float 声明浮点型变量或函数

double 声明双精度变量或函数 //在 c51, picc，c430 中, double 和 float 都是单精度数

基本数据类型限定词

short 声明短整型变量或函数

long 声明长整型变量或函数

signed 声明有符号类型变量或函数

unsigned 声明无符号类型变量或函数

const 声明只读变量

register 声明寄存器变量

static 声明静态变量

volatile 说明变量在程序执行中可被隐含地改变





基本数据类型转换

隐形转换 1 参与运算的两个操作数类型不同时：字节少的类型向字节多的类型转换，有符号类型向无符号类型转换

2 等号两边的数据类型不同时：右边的数据类型向等号左边的数据类型转换

强制转换 (数据类型) 变量; // 括号后面的“变量”的数据类型，将被强制转换为括号里面的“数据类型”。

结构型数据类型

数组

说明： 数据类型 数组名[矢量长度];

数据类型 数组名[行长度] [列长度];

数据类型 数组名[] = { 数据1, 数据2,... }; //数组赋初值

数据类型 数组名[][] = { {数据11, 数据12,...}, {数据21, 数据22,...}, ... }; //数组赋初值

指针

说明： 数据类型 \*指针; //普通指针

数据类型 (\*指针)(); //函数指针，指向函数的指针

数据类型 (\*指针)[]; //数组指针，指向数组的指针

数据类型 \*函数名(); //指针函数，返回指针的函数

数据类型 \*数组名[]; //指针数组，内容为指针的数组

数据类型 (\*数组名[])(); //函数指针数组

char \*指针 = “字符串”; //字符串变量赋初值

赋值： 指针 = 指针 | 数组名 | (数据类型 \*)地址 | &变量;

\*指针 = 变量 | 值;

运算： 变量 = \*指针; //取“指针”指向的单元的内容

指针 = &变量; //取“变量”的地址

指针++ | 指针--; //“指针”指向“下一个 | 上一个”变量单元

指针 = 指针 + n | 指针 - n; //“指针”指向“下 n 个 | 上 n 个”变量单元

结构

结构说明： 1 struct 结构名

{

成员说明语句;

}

2 typedef struct

{

成员说明语句;

} 结构名;

结构的位域说明： struct 结构名

{

unsegned 成员1 : 位长度;

unsegned 成员2 : 位长度;

...; //成员为无符号数

}

结构变量说明： 1 struct [结构名]

{

成员说明语句;

} 结构变量;

2 struct 结构名 结构变量;

结构成员引用： 结构变量非指针：结构变量.成员;

结构变量是指针：结构变量->成员; 或

(\*结构变量).成员;

联合

联合说明： 1 union 联合名

{

成员说明语句;

}

2 typedef union

{

成员说明语句;

} 联合名;

联合变量说明： 1 union 联合名

{

成员说明语句;

} 联合变量;

2 union 联合名 联合变量;

联合成员引用： 联合变量非指针:联合变量.成员;

联合变量是指针: 联合变量->成员; 或

(\*联合变量).成员;

枚举

枚举说明： enum 枚举名

{

元素1, 元素2,..., 元素n

};

枚举变量说明： 1 enum 枚举名

{

元素1, 元素2,..., 元素n

} 枚举变量;

2 enum 枚举名 枚举变量;

枚举变量用法： 枚举变量 = 枚举值;

流程控制语句

(1) 复合语句

{

语句1;

语句2;

...; //多个语句

}

(2) 分枝语句

① if(条件)

语句;

② if(条件)

语句;

else

语句;

③ if(条件0)

语句;

else if(条件1)

语句;

else if(条件2)

语句;

...; // 多个else if语句

else

语句;

④ switch(表达式)

{

case 常量1: 语句; break;

case 常量2: 语句: break;

...; // 多个case语句

default: 语句;

}

(3) 循环语句

① for([i赋值]; [循环条件]; [i修改])

语句;

② while(条件)

语句;

③ do

语句;

while(条件);

(4) 转向语句

break; //跳出循环语句

continue; //跳过循环语句的后续语句

goto 标号; //跳到用“标号”标识的语句行

函数

函数说明：数据类型 函数名(形式参数); //若被调用的函数定义在调用函数之前，可以不进行说明

extern 数据类型 函数名(形式参数); //外部函数，即在另一个 c 文件中定义的，用extern说明

函数定义：数据类型 函数名(形式参数)

{

变量说明;

可执行的语句;

[return [返回值];]

}

函数调用：函数名(实际参数); //运行“函数”

变量 = 函数名(实际参数); //运行“函数”和取返回值

头文件

1 \*assert.h 断言宏

2 ctype.h 字符处理函数

3 \*errno.h 错误定义

4 \*float.h 浮点定义

5 \*limits.h 极限定义

6 local.h 地区化处理函数

7 math.h 数学函数

8 setjmp.h 全局跳转函数

9 signal.h 信号处理函数

10 stdarg.h 可变参数处理函数

11 \*stddef.h 公共定义

12 stdio.h 输入输出函数

13 stdlib.h 实用工具函数

14 string.h 字符串处理函数

15 time.h 日期和时间函数

\* 这5个头文件中不包含函数声明

附录

预处理命令

assert() assert()是与预处理相关的宏函数，用于程序调试

#if #if...#endif，#if... #elif...，

#else...#endif

#ifdef #ifdef...#endif，#ifndef...#endif

#ifndef

#else

#elif

#endif

#define #define... #undef

#undef

#include

#error

#line

#pragma

预定义宏

\_\_LINE\_\_ 包含正在编译的程序的行号

\_\_FILE\_\_ 包含正在编译的程序的文件名

\_\_DATE\_\_ 包含形如month/day/year(月/日/年)的串，代表源文件翻译成目标码的日期

\_\_TIME\_\_ 包含形如hour:minute:second(时：分：秒) 的串，代表源代码编译成目标码的时间

\_\_STDC\_\_ 如果\_\_STDC\_\_的内容是十进制常数1，则表示编译程序的实现符合标准C

包含文件的说明

#include <stdio.h> 标准头文件，在系统目录下

#include "crc\_new32.h" 自定义头文件，在当前目录中

Printf用法举例

在单片机c语言程序中，printf函数可用于在终端窗口中输出结果

printf("%d,%o\n",a,a); 以十进制和八进制输出数a并回车

printf("0x%08lx, 0x%04lx ",a,b); 以十六进制输出长整数a和整数b，左端加0x，不足8位和4位宽度，左端用0 补齐

printf("%.4f",a); 以 4位小数输出浮点数a

printf("%s",a); 输出字符串a

库函数

[字符处理函数](#字符串处理函数) [地区化处理函数](#地区化处理函数) [数学函数](#数学函数) [全局跳转函数](#全局跳转函数) [信号处理函数](#信号处理函数)

[可变参数处理宏](#可变参数处理宏) [输入输出函数](#输入输出函数) [实用工具函数](#实用工具函数)  [字符串处理函数](#字符串处理函数) [日期和时间函数](#日期和时间函数函数)

各个头文件之下，仅列出了所包含的函数的名称和用途，关于函数的详细说明，请参阅国家标准 GB/T 15272-94 程序设计语言 C。如果在上网，可进入 <http://jpkc.whu.edu.cn/jpkc2005/alprogram/jpkc/ckzl/zl/TC20%BF%E2%BA%AF%CA%FD/ansicfunction.htm>，查看“ANSI C 标准函数库”。

字符处理函数 ctype.h

本类别函数用于对单个字符进行处理，包括字符的类别测试和字符的大小写转换。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 字符测试函数 | 是否字母和数字 | isalnum |
| 是否字母 | isalpha |
| 是否控制字符 | iscntrl |
| 是否数字 | isdigit |
| 是否可显示字符（除空格外） | isgraph |
| 是否小写字母 | islower |
| 是否可显示字符（包括空格） | isprint |
| 是否既不是空格，又不是字母和数字的可显示字符 | ispunct |
| 是否空格 | isspace |
| 是否大写字母 | isupper |
| 是否16进制数字（0－9，A-F）字符 | isxdigit |
| 大小写转换函数 | 转换为大写字母 | toupper |
| 转换为小写字母 | tolower |

地区化处理函数 local.h

本类别的函数用于处理不同国家的语言差异。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 地区控制函数 | 地区设置 | \*setlocale 指针函数 |
| 数字格式约定查询函数 | 国家的货币、日期、时间等的格式转换 | localeconv |

数学函数 math.h

本分类给出了各种数学计算函数，必须提醒的是 ansi c 标准中的数据格式并不符合 ieee754 标准，但一些 c 语言编译器却遵循ieee754 （例如frinklin c51）。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 三角函数 | 反余弦 | acos |
| 反正弦 | asin |
| 反正切 | atan |
| 反正切2 | atan2 |
| 余弦 | cos |
| 正弦 | sin |
| 正切 | tan |
| 双曲函数 | 双曲余弦 | cosh |
| 双曲正弦 | sinh |
| 双曲正切 | tanh |
| 指数和对数函数 | 指数函数 | exp |
| 指数分解函数 | frexp |
| 乘积指数函数 | ldexp |
| 自然对数 | log |
| 以 10 为底的对数 | log10 |
| 浮点数分解函数 | modf |
| 幂函数 | 幂函数 | pow |
| 平方根函数 | sqrt |
| 最近整数，绝对值，余数函数 | 最小整数 | ceil |
| 最大整数 | floor |
| 绝对值 | fabs |
| 余数 | fmod |

全局跳转函数 setjmp.h

本分类函数用于实现在不同底函数之间直接跳转代码。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 保存调用环境宏 |  | setjmp |
| 恢复调用环境函数 |  | longjmp |

信号处理函数 signal.h

本分类函数用于处理那些在程序执行过程中发生例外的情况。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 指定信号处理函数 |  | signal |
| 发送信号函数 |  | raise |

可变参数处理宏 stdarg.h

本类函数用于实现诸如printf,scanf等参数数量可变底函数。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 可变参数开始宏 |  | va\_start |
| 可变参数结束宏 |  | a\_end |
| 访问下一个可变参数宏 |  | va\_arg |

输入输出函数 stdio.h

本分类用于处理包括文件、控制台等各种输入输出设备，各种函数以“流”的方式实现。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 文件操作函数 | 删除文件 | remove |
| 修改文件名称 | rename |
| 生成临时文件名称 | \*tmpfile 指针函数 |
| 得到临时文件路径 | tmpnam |
| 文件访问函数 | 关闭文件 | fclose |
| 刷新缓冲区 | fflush |
| 打开文件 | fopen |
| 将已存在的流指针和新文件连接 | freopen |
| 设置磁盘缓冲区 | setbuf |
| 设置磁盘缓冲区 | setvbuf |
| 格式化输入与输出函数 | 格式输出 | fprintf |
| 格式输出（控制台） | printf |
| 格式输入 | fscanf |
| 格式输入（控制台） | scanf |
| 格式输出到缓冲区 | sprintf |
| 从缓冲区中按格式输入 | sscanf |
| 格式化输出 | vfprintf |
| vprintf |
| vsprintf |
| 字符输入输出函数 | 字符输入 | fgetc |
| fputc |
| 字符串输入 | fgets |
| fputs |
| 字符输入（控制台） | getc |
| getchar |
| 字符输出（控制台） | putc |
| putchar |
| 字符串输入（控制台） | gets |
| puts |
| 字符输出到流的头部 | ungetc |
| 直接输入输出函数 | 直接流读操作 | fread |
| 直接流写操作 | fwrite |
| 文件定位函数 | 得到文件位置 | fgetpos |
| 文件位置移动 | fseek |
| 文件位置设置 | fsetpos |
| 得到文件位置 | ftell |
| 文件位置复零位 | rewind |
| 错误处理函数 | 错误清除 | clearerr |
| 文件结尾判断 | feof |
| 文件错误检测 | ferror |
| 得到错误提示字符串 | perror |

实用工具函数 stdlib.h

本分类给出了一些函数无法按以上分类，但又是编程所必须要的。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 字符串转换函数 | 字符串转换为整数 | atoi |
| 字符串转换为长整数 | atol |
| 字符串转换为浮点数 | strtod |
| 字符串转换为长整数 | strtol |
| 字符串转换为无符号长整型 | strtoul |
| 伪随机序列产生函数 | 产生随机数 | rand |
| 设置随机函数的起动数值 | srand |
| 存储管理函数 | 分配存储器 | calloc |
| 释放存储器 | free |
| 存储器分配 | malloc |
| 重新分配存储器 | realloc |
| 环境通信函数 | 中止程序 | abort |
| 退出程序执行，并清除环境变量 | atexit |
| 退出程序执行 | exit |
| 读取环境参数 | getenv |
| 程序挂起，临时执行一个其他程序 | system |
| 搜索和排序函数 | 二分查找（数据必须已排序） | bsearch |
| 快速排序 | qsort |
| 整数运算函数 | 求绝对值 | abs |
| 得到除法运算的商和余数 | div |
| 求长整形底绝对值 | labs |
| 求长整形除法的商和余数 | ldiv |
| 多字节字符函数 | 得到多字节字符的字节数 | mblen |
| 把多字节字符转换为宽字符，返回多字节字符的字节数 | mbtowc |
| 把宽字符转换为多字节字符，返回宽字节的字节数 | wctomb |
| 多字节字符串函数 | 把多字节字符串转换为宽字符串，返回修改过的数组元素数目 | mbstowcs |
| 把宽字符串转换为多字节字符串组，返回修改过的字节数 | wcstombs |

字符串处理函数 string.h

本分类的函数用于对字符串进行合并、比较等操作。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 字符串拷贝 | 块拷贝（目的和源存储区不可重叠） | memcpy |
| 块拷贝（目的和源存储区可重叠） | memmove |
| 串拷贝 | strcpy |
| 按长度的串拷贝 | strncpy |
| 字符串连接函数 | 串连接 | strcat |
| 按长度连接字符串 | strncat |
| 字符串比较函数 | 块比较 | memcmp |
| 字符串比较 | strcmp |
| 字符串比较（用于非英文字符） | strcoll |
| 按长度对字符串比较 | strncmp |
| 字符串转换 | strxfrm |
| 字符与字符串查找 | 字符查找 | memchr |
| strchr |
| 字符串查找 | strcspn |
| strpbrk |
| strrchr |
| strspn |
| strstr |
| 字符串分解 | strtok |
| 其它函数 | 字符串设置 | memset |
| 错误字符串映射 | strerror |
| 求字符串长度 | strlen |

日期和时间函数 time.h

本类别给出时间和日期处理函数。

|  |  |  |
| --- | --- | --- |
| 函数类别 | 函数用途 | 函数名 |
| 时间操作函数 | 得到处理器时间 | clock |
| 得到时间差 | difftime |
| 设置时间 | mktime |
| 得到时间 | time |
| 时间转换函数 | 得到以ASCII码表示的时间 | asctime |
| 得到字符串表示的时间 | ctime |
| 将日历时间转换为国际标准时间 | gmtime |
| 将日历时间转换为当地时间 | localtime |
| 得到指定格式的时间 | strftime |
|  | | |

c51 和 picc 的特点

1 float，double 存储格式

keil c51 ieee754 D32，不论类型声明为double还是float，都是D32格式

Hi-Tech picc ieee754 D24，D32，声明为double类型，编译前，设置为D32格式或D24格式

2 ROM 数据表

keil c51 用 code 标识在程序存储器中存放

Hi-Tech picc 用 const 标识在程序存储器中存放

注：在 ROM 中存放数组数据和结构数据，还可以把结构作为数组元素，这样就形成了一个简单的数据库。

3 中断函数标识

keil c51 void timer0 (void) interrupt i [using j] {;} /i = 0,1,2,… j = 0,1,2,3

Hi-Tech picc void interrupt tc\_int(void){;}

4 c 语言与汇编语言混合编程

方法1 在 c 语言中嵌入汇编语言指令：嵌入格式见下面的举例。

方法2 调用汇编语言函数：① 在工程的选项中设置, 使能生成汇编语言文件，② 建立一个独立的 .c 文件，例如 sub.c，将 sub.c 加入到工程中，在 sub.c 中写一个 c 函数框架，在工程中的其它函数中调用，③ make 工程(也可单独 compile 这个 .c 文件)，假定从 sub.c 生成了汇编语言文件 sub.asm，对于 msp430 是 .s43，④ 修改 sub.asm 中的汇编语言代码，⑤ 从工程中移除 sub.c，将修改过的 sub.asm 文件添加到工程中，⑥ 再编译。

在 picc 中嵌入汇编语言语句：

#asm

BCF \_STATUS, 6

BSF \_STATUS, 5 // 指向BANK1

MOVLW 0xab

MOVWF \_TRISB // 以上 STATUS，TRISB 都是头文件中定义的符号，也可为全局变量

#endasm

asm("RLF \_var, f"); // 另一种格式，var 为全局变量或头文件中定义的符号

在 c51 中嵌入汇编语言语句：

#pragma asm

mov 目标, #0x20 //“目标”= 51 cpu 符号，c 全局变量

#pragma endasm

在 Project 窗口中，选中包含汇编代码的 C 文件，右键，快捷菜单 / Options for ...：“Options for ...”对话框：选中“propertice”选项卡：选中“Generate Assembler SRC File”和“Assemble SRC File”多选项，使检查框由灰色变成黑色(有效)状态。

在 c430 中嵌入汇编语言语句：

asm ("mov.w #0x2a78, 目标"); //“目标”= 430 cpu 符号， c 全局变量

C语言发展简史

起源

C 语言最早的原型是 ALGOL 60，1963 年，剑桥大学将其发展成为 CPL（Combined Programing Language）。1967 年，剑桥大学的 Matin Richards 对 CPL 语言进行了简化，产生了 BCPL 语言。1970 年，美国贝尔实验室（Bell Labs）的 Ken Thompson 将 BCPL 进行了修改，并取名叫做 B 语言，意思是提取 CPL 的精华（Boiling CPL down to its basic good features），并用 B 语言写了第一个 UNIX 系统。1973 年，AT&T 贝尔实验室的 Dennis Ritchie（D.M.RITCHIE）在 BCPL 和 B 语言的基础上设计出了一种新的语言，取 BCPL 中的第二个字母为名，这就是大名鼎鼎的 C 语言。随后不久，UNIX 的内核（Kernel）和应用程序全部用 C 语言改写。从此，C 语言成为 UNIX 环境下使用最广泛的主流编程语言。

K&R C

1978 年，Dennis Ritchie 和 Brian Kernighan 合作推出了《The C Programming Language》的第一版（按照惯例，经典著作一定有简称，该著作简称为 K&R），书末的参考指南 （Reference Manual） 一节给出了当时 C 语言的完整定义，成为那时 C 语言事实上的标准，人们称之为 K&R C，从这一年以后，C 语言被移植到了各种机型上，并受到了广泛的支持，使 C 语言在当时的软件开发中几乎一统天下。

C89 （ANSI C）

随着 C 语言在多个领域的推广和应用，一些新的特性不断被各种编译器实现并添加进来，于是，建立一个新的“无歧义、与具体平台无关的 C 语言定义” 成为越来越重要的事情。1983 年，ASC X3（ANSI 属下专门负责信息技术标准化的机构，现已改名为 INCITS）成立了一个专门的技术委员会 J11（J11 是委员会编号，全称是 X3J11），负责起草关于 C 语言的标准草案。1989 年，草案被 ANSI正式通过，成为美国国家标准，被称为 C89 标准。（ANSI 是美国国家标准协会的缩写）

C90 （ISO C）

随后，《The C Programming Language》第二版开始出版发行，书中内容根据 ANSI C（C89）进行了更新。1990 年，在 ISO/IEC JTC1/SC22/WG14 （ISO/IEC 联合技术第 I 委员会第 22 分委员会第 14 工作组） 的努力下，ISO 批准了 ANSI C 成为国际标准，于是 ISO C（又称为 C90） 诞生了。除了标准文档在印刷编排上的某些细节不同外，ISO C（C90） 和 ANSI C（C89） 在技术上完全一样。 C95

之后，ISO 在 1994、1996 年 分别出版了 C90 的技术勘误文档，更正了一些印刷错误，并在1995 年通过了一份 C90 的技术补充，对 C90 进行了微小的扩充，经过扩充后的 ISO C 被称为 C95。

C99

1999 年，ANSI 和 ISO 又通过了最新版本的 C 语言标准和技术勘误文档，该标准被称为 C99。这基本上是目前关于 C 语言的最新，最权威的定义了。

现在，各种 C 编译器都提供了 C89（C90） 的完整支持，对 C99 还只提供了部分支持，还有一部分提供了对某些 K&R C 风格的支持。

目前主要的C语言规范有C89（C90），C95（94）和C99。C89是最早的C语言规范，于89年提出，90年先由美国国家标准局推出ANSI版本，后来被接纳为ISO国际标准 （ISO/IEC 9899:1990），因而有时也称为c90。但在94和96年分别对c90进行了两次错误修正。gcc支持的是修正后的C89（90）版本的C语言规范。在95年提出过对90版规范的修订案，称为 C95或者AMD1。gcc也支持C95规范。最新的一次C规范修订在99年制定（ISO/IEC 9899:1999），即常称的C99规范。在2001年对C99的错误进行了修正，gcc支持的修正后的c99规范，但是到目前为止，gcc还没有完成对c99规范的完全支持。