GCC 扩展

Statement Exprs

使用圆括号和花括号，可以将 statement 用作表达式，如

```
({ int y = foo (); int z;
   if (y > 0) z = y;
   else z = - y;
   z; })
```

Local Labels

用于宏函数中的 label

```
#define SEARCH(value, array, target)              \
do {                                              \
  __label__ found;                                \
  typeof (target) _SEARCH_target = (target);      \
  typeof (*(array)) *_SEARCH_array = (array);     \
  int i, j;                                       \
  int value;                                      \
  for (i = 0; i < max; i++)                       \
    for (j = 0; j < max; j++)                     \
      if (_SEARCH_array[i][j] == _SEARCH_target)  \
        { (value) = i; goto found; }              \
  (value) = -1;                                   \
 found:;                                          \
} while (0)
```

Labels as Values

可以把地址传给指针，并跳转到该地址

```
void *ptr;
/* ... */
```

```
ptr = &&foo;
goto *ptr;
```

Nested Functions

在函数中定义的函数

```
foo (double a, double b)
{
  double square (double z) { return z * z; }

  return square (a) + square (b);
}
```

Constructing Function Calls

可以将函数的参数直接其他函数，而不需要知道入参的类型或个数。

该功能对于复杂函数会出错，极少使用。

Typeof

获得表达式的类型

Conditionals with omitted operands

x?:y ==> x?x:y

Long Long

64 位整数

Complex Number

复数

Hex Floats

16 进制浮点数，1.55e1 ==》 0x1.fp3

Zero Length

大小为 0 的数组

```
struct line {
  int length;
  char contents[0];
};
```

zy：VC 使用 pragma 取消告警的方式也可以使用该语法

Variable Length

变长数组

```
struct entry
tester (int len, char data[len][len])
{
  /* ... */
}
```

Empty Structure

空结构

```
struct empty {
};
```

Variadic Macros

变参宏函数

```
#define debug(format, ...) fprintf (stderr, format, __VA_ARGS__)
```

Escaped Newlines

空行的判断标准更加宽松？？

Subscripting

可以返回结构体

```
struct foo {int a[4];};
```

```
struct foo f();

bar (int index)
{
  return f().a[index];
}
```

Pointer Arith

void 指针和函数指针可以运算

Initializers

非常量初始化

```
foo (float f, float g)
{
  float beat_freqs[2] = { f-g, f+g };
  /* ... */
}
```

Compound Literals

static int z[] = (int [3]) {1}; ==》 static int z[] = {1, 0, 0};

Designated Inits

指定初始化

int a[6] = { [4] = 29, [2] = 15 }; ==》 int a[6] = { 0, 0, 15, 0, 29, 0 };

Union cast

union 的强制类型转换

union foo { int i; double d; };

u = (union foo) x ==》 u.i = x

Case Ranges

Switch-case 可以使用范围，注意要有空格

example, write this:

case 1 ... 5:

rather than this:

case 1...5:

Mixed Declarations

类似于 C++，声明不一定要在函数的最开始处

Function Attributes

函数属性

Alias weak	void f () __attribute__ ((weak, alias ("__f")));表示 f 的别名是__f 表示可以被用户函数替换，如果用户也有同名函数的话
Always_inline No_inline	一般来说，编译时优化选项没有打开，Inline 函数不做 inline 处理。该标志表示无论何时都 inline 表明函数不是 inline
Flatten	声明该属性的函数，该函数内调用的函数自动做 Inline 处理。函数是否会被 inline 处理，根据函数的大小和当前的 Inline 参数决定。在 unit-at-a-time 模式非常可靠？？
Cdecl stdcall	Intel 386 架构中，假定函数堆栈空间来传递指针。用于覆盖-mrtd 开关。 两个参数的入栈顺序不同
Const	不允许函数访问全局内存（全局变量）
Constructor Destructor	constructor 属性让函数在 main 函数之前执行 destructor 属性让函数在 Main 或 exit 函数后执行
Deprecated	标识函数被删除，不允许被使用，如果使用则产生告警。用于删除代码时使用。

Fastcall	用于 intel 386，使用寄存器传递参数。多于 2 个的参数存在堆栈中，如果函数是变参则全部通过堆栈传递。
Unused used	表明函数不会被使用 表明函数一定会被使用
Visibility	void __attribute__ ((visibility ("protected"))) f () { /* Do something. */; } 符号可以被隐藏，包括 default\hidden\internal\protected 四种属性
Section("sec_name")	一般来说，函数放在 text 段，该属性可以将函数放在自定义段中
Nonnull(arg_index,...) Noreturn Nothrow	extern void * my_memcpy (void *dest, const void *src, size_t len) __attribute__((nonnull (1, 2))); 表明参数 1 和 2 不能为 NULL，如果不使用，则全部不能为 NULL，主要用于编译器优化。如果编译判断可能为 NULL，则有编译告警 表示函数不可能 return 退出，类似于 exit 函数等。用于编译优化 表示函数不可能抛出异常
Regparm(number)	在 I386 架构中，可以使函数前 number 个参数通过寄存器 eax、edx 和 ecx 传递。其他则通过堆栈传递。（但是对于如 solaris8 等系统中，ELF 使用共享库的函数时，默认采用 Lazy binding 技术，当使用第一次时才通过 Loader 加载，这时上述寄存器会被改写，当然可以在编译时进制 lazy binding 选项）

Function Prototypes

支持老格式的 C 函数声明

C++ Comments

支持 C++的注释//

Dollar Signs

可以使用$用于变量声明

Character escapes

可以使用\e 标识<ESC>

Variable Attributes

Align(alignment) Packed	指明变量和结构的对齐大小，单位 bytes 表示紧缩，1 字节对齐
Cleanup(function)	表示变量离开作用域后，调用此函数
Common Uncommon	分配在默认的数据区 直接分配内存，使用堆
Deprecated	同函数属性，表示被删除，不可使用
Section("sec_name")	变量默认存在.data 或.bss 段，该属性表示变量放在指定的区域中。
Shared	与 section 一同使用，在 windows 中使用，表示变量共享
Tls_model	支持 global-dynamic, local-dynamic, initial-exec 和 local-exec。用于 TLS

Type Attributes

Aligned Packed	同变量，用于类型声明

Alignment

__alignof__可以获知 type 或 varialbe 的对齐

Inline

可以定义 Inline 函数

Extended Asm

后面有详细介绍

Constraints

关于汇编的限制

ASM Labels

使用 asm 关键字，可以将 C 中声明的变量或函数用于汇编

Explicit Reg Vars

在一些架构中，可以使用 register 关键字声明某些变量在全局寄存器或本地寄存器中

Alternative Keywords

不同标准的 C 中，有些使用__asm__来替代 asm 关键字等等

Incomplete Enums

可以不定义枚举的具体内容却使用枚举。这样的枚举不能用于声明变量，但可用于指针。

Function Names

__func__表示当前函数的名称

Return Address

void * __builtin_return_address (unsigned int level)

指定函数的返回地址，level 表示相对于当前函数的调用层数

void * __builtin_frame_address (unsigned int level)

指定函数的 frame 地址，类似于上一个函数。frame 指参数后的地址，第一行执行代码为帧

Vector Extensions

矢量计算的操作

Offsetof

__builtin_offsetof(type, name)，计算结构成员的偏移，可以用其他宏代替

Atomic builtins

用于 intel itanium 处理器的部分指令

Object Size checking

为防止缓冲区溢出攻击设计的 size_t __builtin_object_size (void * ptr, int type)

```
struct V { char buf1[10]; int b; char buf2[10]; } var;
char *p = &var.buf1[1], *q = &var.b;

/* Here the object p points to is var.  */
assert (__builtin_object_size (p, 0) == sizeof (var) - 1);
/* The subobject p points to is var.buf1.  */
assert (__builtin_object_size (p, 1) == sizeof (var.buf1) - 1);
/* The object q points to is var.  */
assert (__builtin_object_size (q, 0)
      == (char *) (&var + 1) - (char *) &var.b);
/* The subobject q points to is var.b.  */
assert (__builtin_object_size (q, 1) == sizeof (var.b));
```

type 取值为 0 到 3，分别代表高 bit 和低 bit 是否被设置

Other builtins

void __builtin_prefetch (const void *addr, ...)

将指定数据预读到 CPU 缓存中，提高命中率

Target builtins

针对不同架构的内置函数

Pragmas

针对不同系统或架构的编译选项。如针对 windows 的 pragma pack()

Unamed fields

```
struct {
  int a;
  union {
    int b;
    float c;
  };
  int d;
} foo;
```

可以直接使用 foo.b

TLS

该机制可以使变量为每个线程各分配一个，使用__thread 关键字