**C99可变长数组VLA详解**

转载 2014年10月12日 11:27:03

* 标签：
* [c](http://so.csdn.net/so/search/s.do?q=c&t=blog)
* 414

C90及C++的数组对象定义是静态联编的，在编译期就必须给定对象的完整信息。但在程序设计过程中，我们常常遇到需要根据上下文环境来定义数组的情况，在运行期才能确知数组的长度。对于这种情况，C90及C++没有什么很好的办法去解决（STL的方法除外），只能在堆中创建一个内存映像与需求数组一样的替代品，这种替代品不具有数组类型，这是一个遗憾。C99的可变长数组为这个问题提供了一个部分解决方案。

可变长数组（variable length array，简称VLA）中的可变长指的是编译期可变，数组定义时其长度可为整数类型的表达式，不再象C90/C++那样必须是整数常量表达式。在C99中可如下定义数组：

int n = 10, m = 20;

char a[n];

int b[m][n];

a的类型为char[n]，等效指针类型是char\*，b的类型为int[m][n]，等效指针类型是int(\*)[n]。int(\*)[n]是一个指向VLA的指针，是由int[n]派生而来的指针类型。

由此，C99引入了一个新概念：可变改类型（variably modified type，简称VM）。一个含有源自VLA派生的完整声明器被称为可变改的。VM包含了VLA和指向VLA的指针，注意VM类型并没有创建新的类型种类，VLA和指向VLA的指针仍然属于数组类型和指针类型，是数组类型和指针类型的扩展。

一个VM实体的声明或定义，必须符合如下三个条件：

1。代表该对象的标识符属于普通标识符（ordinary identifier）；

2。具有代码块作用域或函数原型作用域；

3。无链接性。

Ordinary identifier指的是除下列三种情况之外的标识符：

1。标签（label）；

2。结构、联合和枚举标记（struct tag、uion tag、enum tag）；

3。结构、联合成员标识符。

这意味着VM类型的实体不能作为结构、联合的成员。第二个条件限制了VM不能具有文件作用域，存储连续性只能为auto，这是因为编译器通常把全局对象存放于数据段，对象的完整信息必须在编译期内确定。

VLA不能具有静态存储周期，但指向VLA的指针可以。

两个VLA数组的相容性，除了满足要具有相容的元素类型外，决定两个数组大小的表达式的值也要相等，否则行为是未定义的。

下面举些实例来对数种VM类型的合法性进行说明：

#include <stdio.h>

int n = 10;

int a[n];        /\*非法，VM类型不能具有文件作用域\*/

int (\*p)[n];      /\*非法，VM类型不能具有文件作用域\*/

struct test

{

       int k;

       int a[n];     /\*非法，a不是普通标识符\*/

       int (\*p)[n];   /\*非法，p不是普通标识符\*/

};

int main( void )

{

       int m = 20;

       struct test1

       {

              int k;

              int a[n];         /\*非法，a不是普通标识符\*/

              int (\*p)[n];       /\*非法，a不是普通标识符\*/

       };

       extern int a[n];       /\*非法，VLA不能具有链接性\*/

       static int b[n];        /\*非法，VLA不能具有静态存储周期\*/

       int c[n];             /\*合法，自动VLA\*/

       int d[m][n];          /\*合法，自动VLA\*/

       static int (\*p1)[n] = d;  /\*合法，静态VM指针\*/

       n = 20;

       static int (\*p2)[n] = d;  /\*未定义行为\*/

       return 0;

}

一个VLA对象的大小在其生存期内不可改变，即使决定其大小的表达式的值在对象定义之后发生了改变。有些人看见可变长几个字就联想到VLA数组在生存期内可自由改变大小，这是误解。VLA只是编译期可变，一旦定义就不能改变，不是运行期可变，运行期可变的数组叫动态数组，动态数组在理论上是可以实现的，但付出的代价可能太大，得不偿失。考虑如下代码：

#include <stdio.h>

int main( void )

{

       int n = 10, m = 20;

       char a[m][n];

       char (\*p)[n] = a;

       printf( “%u %u”, sizeof( a ), sizeof( \*p ) );

       n = 20;

       m = 30;

       printf( “/n” );

       printf( “%u %u”, sizeof( a ), sizeof( \*p ) );

       return 0;

}

虽然n和m的值在随后的代码中被改变，但a和p所指向对象的大小不会发生变化。

上述代码使用了运算符sizeof，在C90/C++中，sizeof从操作数的类型去推演结果，不对操作数进行实际的计算，运算符的结果为整数常量。当sizeof的操作数是VLA时，情形就不同了。sizeof必须对VLA进行计算才能得出VLA的大小，运算结果为整数，不是整数常量。

VM除了可以作为自动对象外，还可以作为函数的形参。作为形参的VLA，与非VLA数组一样，会调整为与之等效的指针，例如：

void func( int a[m][n] ); 等效于void func( int (\*a)[n] );

在函数原型声明中，VLA形参可以使用\*标记，\*用于[]中，表示此处声明的是一个VLA对象。如果函数原型声明中的VLA使用的是长度表达式，该表达式会被忽略，就像使用了\*标记一样，下面几个函数原型声明是一样的：

void func( int a[m][n] );

void func( int a[\*][n] );

void func( int a[ ][n] );

void func( int a[\*][\*] );

void func( int a[ ][\*] );

void func( int (\*a)[\*] );

\*标记只能用在函数原型声明中。再举个例：

#include<stdio.h>

void func( int, int, int a[\*][\*] );

int main(void)

{

       int m = 10, n = 20;

       int a[m][n];

       int b[m][m\*n];

       func( m, n, a );     /\*未定义行为\*/

       func( m, n, b );

    return 0;

}

void func( int m, int n, int a[m][m\*n] )

{

       printf( "%u/n", sizeof( \*a ) );

}

除了\*标记外，形参中的数组还可以使用类型限定词const、volatile、restrict和static关键字。由于形参中的VLA被自动调整为等效的指针，因此这些类型限定词实际上限定的是一个指针，例如：

void func( int, int, int a[const][\*] );

等效于

void func( int, int, int ( \*const a )[\*] );

它指出a是一个const对象，不能在func内部直接通过a修改其代表的对象。例如：

void func( int, int, int a[const][\*] );

……..

void func( int m, int n, int a[const m][n] )

{

       int b[m][n];

       a = b;        /\*错误，不能通过a修改其代表的对象\*/

}

       static表示传入的实参的值至少要跟其所修饰的长度表达式的值一样大。例如：

void func( int, int, int a[const static 20][\*] );

……

int m = 20, n = 10;

int a[m][n];

int b[n][m];

func( m, n, a );

func( m, n, b );     /\*错误，b的第一维长度小于static 20\*/

       类型限定词和static关键字只能用于具有数组类型的函数形参的第一维中。这里的用词是数组类型，意味着它们不仅能用于VLA，也能用于一般数组形参。

       总的来说，VLA虽然定义时长度可变，但还不是动态数组，在运行期内不能再改变，受制于其它因素，它只是提供了一个部分解决方案。