不安全代码：代码的执行不受公共语言运行时（CLR）的完全管理。

在C#中，可以在不安全代码中使用指针。

unsafe和fixed修饰符

unsafe标记不安全代码，fixed修饰符防止垃圾回收器移动托管变量，只能用在unsafe标记的不安全上下文中。

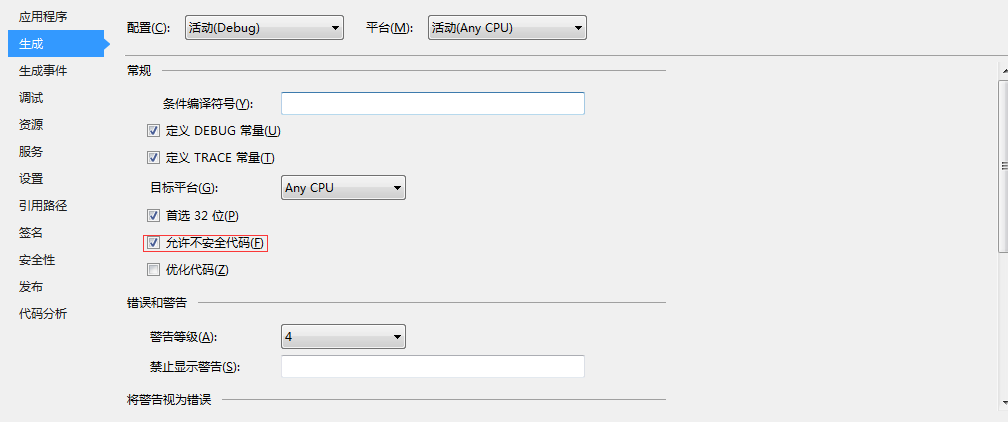
fixed (type\* p = &fixedObj)

{

// use fixed object

}

在VS2013中，使用unsafe关键字，需要设置工程的属性：



例：程序unsafe\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// unsafe使用：指针

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace unsafe\_test1

{

class Test

{

public int num;

public Test(int i) { num = i; }

}

class PointerTest

{

// 不安全代码段

unsafe static void Main(string[] args)

{

Test obj = new Test(19);

// fixed防止垃圾回收器移动obj.num

// 直到fixed代码段结束

fixed (int\* p = &obj.num)

{

*Console*.*WriteLine*("Initial vale of num is " + obj.num);

\*p = 10;

*Console*.*WriteLine*("New value of num is " + obj.num);

}

}

}

}

输出结果为：

Initial vale of num is 19

New value of num is 10

C#中，指针只能操作值类型。

通过指针访问结构成员：结构中不能包含引用类型。

例：程序unsafe\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 不安全代码：指针访问结构体成员

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace unsafe\_test2

{

struct MyStruct

{

public int a;

public int b;

// 结构中如果包含引用类型，指针无法获取其地址

// public string name;

public int Sum() { return a + b; }

}

class PointerTest

{

static void Main(string[] args)

{

MyStruct obj = new MyStruct();

unsafe

{

MyStruct\* p = &obj;

p->a = 10;

p->b = 20;

*Console*.*WriteLine*("Sum is " + p->Sum());

}

}

}

}

在计算机中，内存中的每一个地址都代表一个字节。

指针运算：有且只有4种算术运算符可以用于指针：++、--、+和-。

不能将void\*指针用于指针的算法运算。

指针之间不能进行+运算，但可以想减

C#中指针的算术运算和C/C++中的完全一致。

指针+或-的操作数只能为整数，不能是浮点数，也不能是decimal。

例：程序unsafe\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 指针的运算

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace unsafe\_test3

{

class PointerTest

{

static void Main(string[] args)

{

int n = 2;

int m = 3;

int k = 20;

unsafe

{

int\* n\_ptr = &n;

// 将指针指向的地址显示成十六进制

*Console*.*WriteLine*("original n\_ptr address is {0:X4}", (int)n\_ptr);

++n\_ptr;

*Console*.*WriteLine*("++n\_ptr address is {0:X4}", (int)n\_ptr);

--n\_ptr;

*Console*.*WriteLine*("--n\_ptr address is {0:X4}", (int)n\_ptr);

}

*Console*.*WriteLine*();

unsafe

{

int\* m\_ptr = &m;

int\* k\_ptr = &k;

*Console*.*WriteLine*("original m\_ptr address is {0:X4}", (int)m\_ptr);

*Console*.*WriteLine*("original k\_ptr address is {0:X4}", (int)k\_ptr);

*Console*.*WriteLine*();

long x = m\_ptr - k\_ptr;

*Console*.*WriteLine*("m\_ptr - k\_ptr is {0}", x);

m\_ptr = m\_ptr + 9;

*Console*.*WriteLine*("m\_ptr + 9 address is {0:X4}", (int)m\_ptr);

m\_ptr = m\_ptr - 5;

*Console*.*WriteLine*("m\_ptr - 5 address is {0:X4}", (int)m\_ptr);

*Console*.*WriteLine*("\*m\_ptr " + \*m\_ptr); // 随机数

void\* v\_ptr = &m;

// ++v\_ptr; // error，void指针不能进行算法运算

// Console.WriteLine("\*v\_ptr " + \*v\_ptr); // error，不能解析void指针

}

}

}

}

输出结果为：

original n\_ptr address is 39ECD8

++n\_ptr address is 39ECDC

--n\_ptr address is 39ECD8

original m\_ptr address is 39ECD4

original k\_ptr address is 39ECD0

m\_ptr - k\_ptr is 1

m\_ptr + 9 address is 39ECF8

m\_ptr - 5 address is 39ECE4

\*m\_ptr 3796224

从输出结果可以看出：

1. n\_ptr的原始地址为39ECD8，执行了++操作后，改变为39ECDC，两个地址相差了4个字节，正好是整型变量的长度。--操作后，结果同样如此。
2. m\_ptr的地址为39ECD4，k\_ptr的地址为39ECD0，两个指针相减后的值为两个指针的地址相减后的值 / 指针指向的类型大小，即两个指针之间存在的指针所指向的类型的元素的个数。
3. 指针与整数执行+或-操作，其过程与执行++或--类似。

在C#中，数组是对象，但可以通过指针来访问数组，与C++非常类似。

在C#中，字符串是对象，但可以通过指针来访问字符串中的单个字符，与C++也类似。

例：程序unsafe\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// 指针访问数组

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace unsafe\_test4

{

class PointerTest

{

static void Main(string[] args)

{

int[] nums = new int[10];

string str = "this is a test.";

unsafe

{

// 访问数组

fixed (int\* n\_ptr = nums)

{

for (int i = 0; i < 10; ++i)

{

n\_ptr[i] = i;

}

for (int i = 0; i < 10; ++i)

*Console*.*WriteLine*("n\_ptr[{0}] : {1}", i, n\_ptr[i]);

}

*Console*.*WriteLine*();

fixed (int\* n\_ptr = nums)

{

for (int i = 0; i < 10; ++i)

{

\*(n\_ptr + i) = i;

}

for (int i = 0; i < 10; ++i)

*Console*.*WriteLine*("\*(n\_ptr + {0}) : {1}", i, \*(n\_ptr + i));

}

*Console*.*WriteLine*();

// 访问字符串

fixed (char\* c\_ptr = str)

{

for (int i = 0; i < str.*Length*; ++i)

{

*Console*.*Write*(c\_ptr[i]);

}

}

*Console*.*WriteLine*();

}

}

}

}

输出结果为：

n\_ptr[0] : 0

n\_ptr[1] : 1

n\_ptr[2] : 2

n\_ptr[3] : 3

n\_ptr[4] : 4

n\_ptr[5] : 5

n\_ptr[6] : 6

n\_ptr[7] : 7

n\_ptr[8] : 8

n\_ptr[9] : 9

\*(n\_ptr + 0) : 0

\*(n\_ptr + 1) : 1

\*(n\_ptr + 2) : 2

\*(n\_ptr + 3) : 3

\*(n\_ptr + 4) : 4

\*(n\_ptr + 5) : 5

\*(n\_ptr + 6) : 6

\*(n\_ptr + 7) : 7

\*(n\_ptr + 8) : 8

\*(n\_ptr + 9) : 9

this is a test.

sizeof运算符：只能用在unsafe代码中

stackalloc可以从栈中分配内存，但只有在初始化局部变量时才能使用它，必须使用在不安全的上下文中。

例：程序unsafe\_test5

// Copyright 2016.刘珅珅

// author：刘珅珅

// sizeof和stackalloc

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace unsafe\_test5

{

class PointerTest

{

static void Main(string[] args)

{

int n = 2;

double d = 8.5;

unsafe

{

int\* n\_ptr = &n;

*Console*.*WriteLine*("n\_ptr size " + sizeof(int\*)); // 4

double\* d\_ptr = &d;

*Console*.*WriteLine*("d\_ptr size " + sizeof(double\*)); // 4

}

*Console*.*WriteLine*();

unsafe

{

// stackalloc有点类似C++中的new

int\* ptrs = stackalloc int[3];

ptrs[0] = 1;

ptrs[1] = 2;

ptrs[2] = 3;

for (int i = 0; i < 3; ++i )

*Console*.*WriteLine*(ptrs[i]);

}

}

}

}

从上例可以看出，C#中，指针的大小貌似也是4个字节。

fixed关键字可以创建一个固定大小的一维数组，称为固定大小的缓冲区，只能用于不安全上下文的编程。

fixed type buf-name[size];

固定缓冲区只能在结构的内部使用。有点类似C++的数组

例：程序unsafe\_test6

// Copyright 2016.刘珅珅

// author：刘珅珅

// 固定缓冲区fixed

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace unsafe\_test6

{

unsafe struct FixedBankRecord

{

int n;

double b;

char c;

// 固定缓冲区

public fixed byte Name[80];

public double Balance;

public long ID;

}

class FixedTest

{

static void Main(string[] args)

{

unsafe

{

// 结果为120

// 与C++中结构体大小的计算比较类似

// 需要填充字节

*Console*.*WriteLine*("Size of FixedBankRecord is " + sizeof(FixedBankRecord));

}

}

}

}