1. C++/CLI特有的基本数据类型：long long（8字节）和unsigned long long（8字节）。在ISO/ANSI C++中，基本类型和类类型是完全不同的；而在C++/CLI中，所有数据都是作为类类型的对象存储的，或者是作为值类类型，或者是作为引用类类型。将基本类型的数据表示为值类类型的对象是C++/CLI的一个重要特性。

|  |  |  |
| --- | --- | --- |
| 基本类型 | 大小（字节） | CLI值类 |
| bool | 1 | System::Boolean |
| char | 1 | System::SByte |
| signed char | 1 | System::SByte |
| unsigned char | 1 | System::Byte |
| short | 2 | System::Int16 |
| unsigned short | 2 | System::UInt16 |
| int | 4 | System::Int32 |
| unsigned int | 4 | System::UInt32 |
| long | 4 | System::Int32 |
| unsigned long | 4 | System::UInt32 |
| long long | 8 | System::Int64 |
| unsigned long long | 8 | System::UInt64 |
| float | 4 | System::Single |
| double | 8 | System::Double |
| long double | 8 | System::Double |
| wchar\_t | 2 | System::Char |

1. Console类表示标准输入和输出流，对应于键盘和命令行窗口中的命令行。WriteLine()函数是C++/CLI函数，定义在System命名空间的Console类中。Console类中的Write()函数和WriteLine()函数基本相同，唯一的区别是它不在指定的输出之后自动写入换行符。
2. C++/CLI特有的功能—格式化输出，例如：

Console::WriteLine(L"There are {0} packages.", packageCount);

Console::WriteLine(L"There are {0} packages weighting {1} pounds.", packageCount, packageWeight);

Console::WriteLine(L"There are {1} packages weighting {0} pounds.", packageWeight, packageCount);

Console::WriteLine(L"There are {0} packages weighting {1:F2} pounds.", packageCount, packageWeight);

在子串{1:F2}中，冒号将标识要选择参数的索引值1和它后面的格式规范F2分隔开。格式规范中的F表示输出应当采用格式“±ddd.dd...”（其中d代表一个数字），2表示想在小数点后面保留两位。

Console::WriteLine(L"There are {0, 3} packages weighting {1, 5:F2} pounds.", packageCount, packageWeight);

Console::WriteLine(L"There are {0, -3} packages weighting {1, -5:F2} pounds.", packageCount, packageWeight);

可以编写格式为{n, w : Axx}的格式规范，其中n是一个索引值，用于选择格式串之后的参数，w是一个可选的字段宽度规范（有符号的整数，如果为正则值在这个字段中将右对齐，如果为负则左对齐，如果值占用的位置数小于w指定的位置数，那么输出将由空格填充，如果值需要的位置数大于w指定的位置数，那么这个宽度规范将被忽略），A是单个字母，指定如何对值进行格式化，xx是可选的一个或两个数字，指定值的精度。

|  |  |
| --- | --- |
| 格式说明符 | 说明 |
| C或c | 把值作为货币量输出 |
| D或d | 把整数作为十进制值输出。如果指定的精度大于位数，那么将在数字的左边填0 |
| E或e | 按照科学记数法输出浮点值，即利用指数。精度值表示在小数点后输出的位数。 |
| F或f | 把浮点值作为形式为±ddd.dd...的定点数输出 |
| G或g | 以最紧凑的形式输出值，这取决于值的类型和是否指定了精度。如没有指定精度，那么将使用默认精度。 |
| N或n | 把值作为定点十进制值输出，必要时，将以每3位为一组用逗号分隔开。 |
| X或x | 把整数作为十六进制值输出。取决于是否指定X或x，将输出小写十六进制数字的大写形式。 |

1. C++/CLI的键盘输入：Console::ReadLine()函数把整行输入作为字符串读取；Console::Read()函数读取单个字符；Console::ReadKey()函数读取所按的键。例如：String^ line = Console::ReadLine();类型名称String后面的字符^表示这是引用String类型对象的句柄。ConsoleKeyInfo keypress = Console::ReadKey(true);ReadKey()函数的参数为true则对应于按键的字符不在命令行上显示，为false则显示。对应于按键的字符储存在keypress.KeyChar中。
2. 在C++/CLI中可以用safe\_cast实现显式类型强制转换（推荐使用），用法同static\_cast。
3. C++/CLI枚举与ISO/ANSI C++的不同：以enum class Suit : int{Clubs = 1, Diamonds = 1, Hearts, Spades};为例：

（1）始终必须利用枚举类型名称限定正在使用的常量，例如：Suit suit = Suit::Clubs;。

（2）在enum关键字后加入class关键字，此处在枚举中定义的常量是对象，而不仅仅是基本类型的值。默认情况下，它们是Int32类型的对象，封装int类型的值，要试图使用一个常量之前必须先把它强制转换成int类型。

（3）由于C++/CLI枚举是一种类类型，因此不能从局部定义它（例如不能在一个函数内定义它），需要在全局作用域中定义它。

1. .Net Framework在Char类内提供了将字符代码转换为大写（Char::ToUpper()）和小写（Char::ToLower()）函数，以及测试某个字母是大写还是小写的IsUpper()和IsLower()函数（返回bool值）。
2. ConsoleKeyInfo类中有3个可访问的属性：属性Key识别被按的是哪个键，属性KeyChar是被按键的Unicode字符代码，属性Modifiers是表示shift、Alt和Ctrl键的ConsoleModifiers常量的按位组合。ConsoleModifiers是在System库中定义的枚举类型，其定义的常量有Alt、Shift和Control。keypress.Modifiers可以得到所有的修改键，因为Modifiers枚举类型使用FlagsAttribute属性定义的，该属性表明这种类型是一组唯一的位标志。这使得该枚举类型的变量可以由若干与在一起的标志位组成，而Write()或WriteLine()函数可以识别并输出各个标志位。
3. C++/CLI提供了一种名为for each的循环：for each(wchar\_t ch in proverb)，其中proverb变量属于被描述为String句柄的String^类型。Char::IsLetter(ch)判断ch是否为字母。
4. CLR的垃圾回收器能够将不再需要的对象删除，并释放它们所占用的内存（CLR记录着每个堆内对象的变量，如果没有任何变量包含某个对象的地址，则该对象就不可能在程序中再次被引用，因此可以将其删除），使用gcnew运算符而非new运算符分配内存，不需要程序员delete。
5. CLR的垃圾回收过程可能需要压缩堆的内存区域，以消除零碎的未用内存块，所以存储在堆内的数据项的地址有可能改变。因此我们不能再可回收垃圾的堆中使用普通的本地C++指针，因为如果指向的数据位置发生变化，则该指针将不再有效。我们需要在垃圾回收器重新安排堆内数据项的位置时，能够更新相应的地址。在CLR中有两种方法可实现：类似于本地C++指针的跟踪句柄和等价于C++引用的跟踪引用。
6. 在CLR堆上分配的变量——其中也包括所有CLR引用类型，都不能在全局作用域中声明。
7. 声明跟踪句柄，例如：int^ value = 99; int result = 2 \* (\*value) + 15;也可以这样写：int^ result = 0; result = 2\* (\*value) + 15;当我们在等号左边使用句柄时，不需要显示解除引用即可用来存储结果。

——————————————————P180