C#

O C# ("C Sharp") é uma nova linguagem de programação desenvolvida pela Microsoft para sua nova plataforma, a ".NET". O C# é baseado no C/C++, mas tem uma grande influência do Delphi, já que ambos foram criados pela mesma pessoa, Anders Hejlsberg. Neste artigo irei analisar as principais características do C# e como elas se comparam com outras linguagens como C++, Java e Delphi.

Mais uma linguagem, de novo?

A Microsoft apresentou recentemente uma nova linguagem, o C#, como a linguagem padrão para programação na sua nova plataforma "ponto net".

Será que já não temos linguagens de programação em quantidade suficiente? Não é possível usar as que já temos, como por exemplo, o C++? Essa é uma pergunta muito comum. O interessante é que as pessoas que perguntam geralmente fazem questão de ter o computador mais moderno que existe, mas sugerem usar uma linguagem do tempo que a CPU mais moderna era o 286 e que é compatível com outra desenvolvida nos tempo do Intel 8008, o "C"!

A resposta é não, não dá para usar o C++ nem qualquer outra linguagem famosa. Da mesma forma que temos hoje computadores mais capazes e sistemas operacionais mais sofisticados, a tecnologia de linguagens também evoluiu. O que era uma linguagem bem equilibrada há 10 anos atrás simplesmente não dá conta do recado hoje. O C++, por exemplo, carrega uma boa dose de entulho do tempo do "C" original "Kernighan & Richie. Por exemplo: até hoje em C++ os "arrays" e ponteiros são equivalentes, embora sejam conceitos muito diferentes; o C++ não tem o conceito de "eventos", uma das grandes idéias do Delphi; não existe nem enumerações nem um tipo lógico "fortes" e incompatíveis com outros tipos. A linguagem sequer tem um tipo string! A lista é longa.

Todas as linguagens importantes têm os seus "pecados", pela simples razão de que as necessidades que as tornavam boas linguagens há 15 anos atrás, as tornam deficientes hoje. Isto vale para o C/C++, COBOL, Pascal, xBase e BASIC. É possível fazer alguns "remendos", mas depois de um certo ponto é melhor começar de novo.

A sintaxe do C# é muito parecida com a do C++. A princípio, quando uma construção do C++ não oferece problemas, ela é usada. Mas o C# modifica bastante o C++ e não tem a pretensão de manter compatibilidade:

- Todas as variáveis e código são declarados no escopo de classes, de forma semelhante ao Java. É possível, contudo, declarar tipos ("structs" e enumerações) fora do escopo de classes. Nem tudo é uma classe...
- Tipagem forte. Enumerações são tipos próprios e incompatíveis com inteiros e com outras enumerações. Existe um tipo lógico (*bool*) incompatível com inteiros. Os tipos intrínsecos são: lógicos, inteiros de vários tipos e com tamanhos pré-definidos (8, 16, 32 e 64 bits, com e sem sinal), ponto flutuante IEEE de 4 e 8 bytes, string e decimal. Só existe um único tipo "char". Tanto o "char" como a "string" armazenam apenas caracteres Unicode (16 bits). O tipo "decimal" é armazenado como uma mantissa binária

de 96 bits e um expoente na base 10, para um total de 128 bits. A precisão do decimal é de 28 ou 29 dígitos decimais. Existe também “structs”.

- Os objetos, “arrays” e “string” são necessariamente alocados dinamicamente no “heap” com o uso do operador “new”.
- O C# inicializa a maioria das variáveis com zero, verifica se uma variável foi inicializada antes de ser usada, se um array foi indexado fora da faixa e se um inteiro teve sua faixa violada.
- Todas as conversões de tipo (“cast”) são validadas em função do tipo real da variável em tempo de execução, sem exceções. Isto é semelhante ao cast “as” do Delphi, mas no Delphi só temos “as” para objetos, enquanto no C# todos os tipos têm “cast” seguro.
- O operador “.” ‘é usado em diversos lugares, quando em C++ seriam usados “.”, “::” e “->”, como no Delphi.
- Existe um outro tipo de loop: “foreach”, usado para varrer todos os elementos de um array ou “container”.
- O “swich” elenca opções mutuamente exclusivas, como no Delphi. O “break” depois de cada opção é obrigatório.
- O único mecanismo de tratamento de erros do C# é a exception. O mecanismo é muito semelhante ao do Delphi.
- Não existem macros, mas existe compilação condicional, como no Delphi.
- Templates não são suportados, pelo menos por enquanto. A Microsoft acena que talvez seja possível criar um mecanismo semelhante aos templates no futuro. De qualquer forma, no C# suporta “reflections”, o que pode substituir templates em algumas situações.
- O C# suporta sobrecarga de funções e de operadores, como o C++, mas não tem argumentos default como o C++ e o Delphi.
- O C# possui operadores de conversão, mas existe uma sintaxe para indicar se a conversão deve ser implícita ou explícita. O construtor não é usado como operador de conversão. As conversões implícitas em C++ são um grande problema, pois acabam permitindo que o compilador inadvertidamente crie uma conversão que não faz sentido.

O processo de compilação

Um conjunto de arquivos-fonte é diretamente compilado em um arquivo executável (.EXE ou .DLL), sem etapas intermediárias. O arquivo executável é chamado de “assembly” e contém não só o código executável como também informação simbólica (“metadata”). A metadata contém as seguintes informações, dentre outras:

- Nomes dos tipos.
- Nomes dos métodos.
- Nomes dos componentes dos tipos (campos, métodos, propriedades, enumerações, etc).

Este esquema é praticamente idêntico ao Delphi usando “runtime packages”:

Orientação a Objeto

O modelo de orientação a objeto tem as seguintes características básicas:

- Podemos declarar “properties”, que funcionam sintaticamente como campos, mas na verdade chamam um par de métodos para atribuir ou receber o valor da “property”. As propriedades podem ser também “indexadas” com um inteiro, funcionando como se fossem “arrays” ou indexadas com uma “string”, quando passam a funcionar como um dicionário. O ambiente de desenvolvimento sabe criar “editores de propriedades” automaticamente para as propriedades. Isto é Delphi puro.
- Herança simples, com um ancestral comum à todos os objetos chamado “System.Object” O ancestral comum concentra funções de criação, comparação, conversão para string e informações de tipo em tempo de execução. A informação de tipo em tempo de execução (“reflections”) joga um papel fundamental no funcionamento geral da biblioteca de classes, tudo como no Delphi. O C++ não tem estes conceitos.
- Embora a herança seja simples, as classes podem implementar várias “interfaces”. Isto traz as vantagens da herança múltipla sem os seus problemas. Uma interface funciona como se fosse uma “classe abstrata”, que possui apenas protótipos de métodos e nenhuma implementação. Como no Delphi e Java. O C++ não tem tais conceitos.
- Os métodos não são a princípio virtuais e devem ser explicitamente declarados como tais com a palavra reservada “virtual”, como no Delphi e C++. Existe um protocolo específico para indicar se um método de classe derivada reimplementa um método virtual (override) ou o torna não-virtual (new).
- Podemos declarar um tipo que é um “ponteiro para método”, chamado “delegate”. Um “delegate” contém a princípio o endereço da função e também do método que a implementa. Todos os eventos, tão importantes para o funcionamento do ambiente de desenvolvimento são “delegates”. Os delegates permitem que uma classe chame métodos em outras sem exigir que esta outra classe seja derivada. Este é um conceito do Delphi (eventos) que não tem nada semelhante em C++ ou Java.
- Todos os objetos são referências e devem ser criados explicitamente com o operador new, de forma semelhante ao Delphi. Não existem “objetos por valor” como no C++.
- A informação de tipos em tempo de execução permite coisas que normalmente as linguagens compiladas não são capazes como: criar um objeto de uma classe dado seu nome como string, atualizar propriedades dados seu nome e valor como strings e chamar métodos dados seu nome e argumentos como strings. O Smalltalk e o Delphi permitem isto. O C++ não tem nada vagamente semelhante. Tanto o ambiente de desenvolvimento como o de execução confiam pesadamente neste mecanismo para funcionarem.
- Existe um mecanismo para herança de formulários, como no Delphi.

Veja o “Hello World” em C# para um aplicativo em modo console:

```
public class Class1
{
  public static int Main(string[] args)
```

```
  {
    System.Console.WriteLine("Alo, Mundo\n");
    return 0;
  }
}
```

Novidades em relação ao Delphi

Pelo exposto acima, podemos ver que a ".NET Framework" e o C# implementam diversos conceitos que foram utilizados anteriormente no Delphi. No entanto, a nova plataforma da Microsoft vai além do Delphi, introduzindo conceitos realmente novos.

A Plataforma é o sistema operacional

O fato de o mesmo ambiente ser usado para as rotinas básicas, bibliotecas e aplicativos traz por si só algumas vantagens:

- É muito fácil misturar código escrito em diferentes linguagens sem a necessidade de rotinas ou DLL intermediárias para "colar" os diversos elementos.
- O tratamento de erros com exception é mais eficiente e robusto: existe um único caminho para os erros percorrerem, sem rotinas intermediárias.
- O gerenciamento de memória pode ser mais eficiente. Eventuais cópias e as realocações de memória podem ser evitadas.
- A segurança é a mesma para todos os aplicativos e não é necessário fazer verificações a mais ao se cruzar do aplicativo para a biblioteca.

Sem ponteiros

Não existem ponteiros na nova plataforma. Isto não quer dizer que não temos a eficiência dos ponteiros: muitos objetos são tratados por referência. As referências são "ponteiros domesticados": embora internamente elas sejam ponteiros, elas são sempre inicializadas e não podem apontar para locais arbitrários de memória.

Boxing

Os objetos oferecem um modelo muito conveniente para lidar com elementos em nossos programas através da abstração proporcionada por propriedades, métodos, eventos e do mecanismo de herança. O problema é que os objetos têm o custo adicional de serem sempre acessados através de ponteiros ("this", "self") e terem que ser criados e destruídos.

Este custo é irrelevante quando estamos lidando com um objeto complexo e pesado como um formulário na tela ou um arquivo em disco. Mas é um custo muito caro para tipos simples como um inteiro, especialmente visto que a CPU consegue lidar com inteiros de maneira muito eficiente.

A plataforma resolve este problema de uma maneira brilhante: existem duas categorias de tipos: por valor e por referência. O tipos por valor podem ser automaticamente convertidos para referências através de um processo chamado "boxing".

Os tipos por valor são:

- bool
- inteiros
- double
- decimal
- enum
- struct

Os tipos por referência são:

- Classes
- Arrays

Ao declararmos um tipo por valor, ele é alocado na pilha e gerenciado "por cópia". Os tipos por referência devem sempre ser alocados com "new", são acessados através de um ponteiro e são desalocados por mecanismo de "garbage collection".

A situação fica interessante ao chamarmos um método de um tipo por valor, por exemplo:

```
int a = 10;
string s = a.ToString();
```

Quando da chamada do método "ToString", o inteiro "a" é "boxed", ou seja: um ponteiro criado e passado para o método "ToString()".

O "boxing" ocorre automaticamente. Podemos efetuar "unboxing", mas de maneira explícita através de um "downcast", como no exemplo a seguir:

```
int a = 10;                  // Declara um inteiro
object o = a;                // Força "boxing"
int b = (int) o;             // Cast para integer
```

Note que o "downcast" é sempre validado em tempo de execução.

Delegates

Os "delegates" são análogos aos eventos do Delphi, mas vão alguns passos além:

- Podem ser encadeados de forma a chamar uma seqüência de métodos.
- Podem chamar métodos "static", não são associados a um objeto.
- A inicialização é mais flexível. Na verdade, o Delphi não provê um mecanismo documentado para criar um "delegate", embora isso possa ser feito.

Garbage Collection

O gerenciamento de memória é totalmente controlado pelo ambiente.

O seu programa aloca memória indiretamente ao criar variáveis com "new", mas nunca a libera diretamente. O sistema de runtime automaticamente libera a memória das variáveis que não são mais referenciadas, em um processo chamado "garbage collection". Este é um ponto meio polêmico: não existe dúvida que um programador pode gerenciar memória melhor. O problema é que os erros causados por mau gerenciamento de memória são muito comuns. É melhor deixar o

computador fazer um serviço não ótimo, mas correto do que deixar os programadores tentarem fazer um serviço ótimo e errarem.É um caso típico do “ótimo brigando com o bom”.

Este esquema facilita bastante a programação e elimina uma grande fonte de erros.

Attributes

Os atributos permitem associar código a métodos em tempo de execução, modificando sua implementação. Muitas das características da plataforma são implementadas como atributos:

- Criação de “webmethods”.
- Chamada a “webmethods”.
- Implementações de classes COM.
- Chamada a objetos COM.

Conclusão

O C# é um C++ “limpo”, com várias boas idéias do Delphi e outras novas, como “boxing”, “delegates”, “garbage collection” e “attributes”. Ela é muito atraente para programadores C/C++ e Delphi que desejam migrar para a plataforma .”NET” da Microsoft.

© Copyright 2000 por Mauro Sant’Anna – Todos os direitos reservados

Mauro Sant’Anna vem desenvolvendo para Windows desde 1991 e componentes desde o Delphi 1.0.. Ministrou palestras em conferências da Borland nos EUA e tem desenvolvido com C# desde julho. Atualmente coordena os cursos na M.A.S. Informática, empresa eleita pelos sócios do “The Club” como fornecedora do melhor treinamento em Delphi no Brasil. A M.A.S. também está disponível para consultoria e desenvolvimento. Veja maiores detalhes em www.mas.com.br.