helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

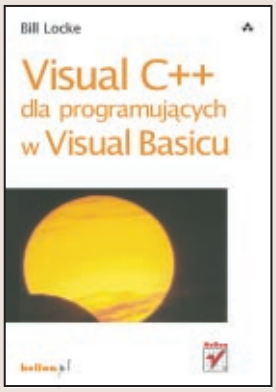
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Visual C++ dla programujących w Visual Basicu



Autor: Bill Locke
Tłumaczenie: Przemysław Steć
ISBN: 83-7197-793-X
Tytuł oryginału: Visual C++ for VB Programmers
Format: B5, stron: 376

C i C++ to języki dla profesjonalnych programistów, którzy nie znoszą ograniczeń. Za ich ogromne możliwości (w C/C++ napisano większość systemów operacyjnych i aplikacji użytkowych, jakie są obecnie wykorzystywane) trzeba jednak zapłacić słoną cenę: nie jest łatwo nauczyć się C/C++, a jeszcze trudniej jest w nich efektywnie programować. Dlatego nie są one polecane jako narzędzia dla początkujących. Skoro jednak — jak sugeruje tytuł tej książki — znasz już Visual Basic, pierwsze kroki masz już za sobą.

Dla chcącego nic trudnego, zwłaszcza, że współczesne, wizualne środowiska programistyczne, takie jak Visual C++ Microsoftu, ułatwiają tworzenie i uruchamianie aplikacji. Dodatkowo, na rynku pojawił się kolejny język z rodziny C: C#. Znosi on wiele trudności, jakie stawiały przed programistą C i C++, stanowiąc jednocześnie podstawowy język w technologii .NET, polecanej przez Microsoft jako przyszłościowy model tworzenia aplikacji, także sieciowych.

C, C++, C# — więcej możliwości, więcej satysfakcji:

- Podstawy języków C i C++
- Różnice między C a Visual Basic
- C++ a Windows
- Pisanie bibliotek DLL
- Komponenty COM, formanty ActiveX, technologia .NET
- Programowanie w języku C#

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

1.

Najpierw trochę historii

W tym rozdziale:

- Historia Visual Basica i C++
- Znaczenie języka Visual Basic
- Znaczenie języka C++
- Język C#

Celem książki jest nauczenie czytelnika programowania w języku C, a ściślej — programowania w językach C, C++ oraz C# w celu rozszerzenia możliwości, jakie oferuje Visual Basic. Z tego względu przedstawiona poniżej historia VB jest ukierunkowana na rozwój narzędzi programistycznych. Zanim rozpoczniemy naszą wyprawę w świat języka C, należy uświadomić sobie, co nas do niej skłoniło. Otóż historia nieraz już pokazała, że każdy programista powinien być elastyczny. Niezależnie od aktualnych tendencji w informatyce czy osobistych preferencji, prędzej czy później pojawia się konieczność zmiany narzędzia programowania i trzeba być na to przygotowanym. Ważne jest, aby w miarę możliwości programować w „najmniejszym wspólnym mianowniku", czyli korzystać z narzędzi uniwersalnych, które umożliwią wykorzystanie stworzonych fragmentów kodu w przyszłości.

Do takich z pewnością należy „czysty" język C. Nawet w kodzie źródłowym nowoczesnych 32-bitowych obiektów sterujących, które autor książki ma w swoim dorobku, występują fragmenty napisane w klasycznym C, które powstały w czasach Visual Basica 1 i 2. Co ważne, w nowych warunkach, poddane niewielkim zmianom, działają równie dobrze jak dawniej w środowisku 16-bitowym.

Rozważmy na przykład programowanie interfejsu graficznego (*GDI – Graphics Device Interface*). Na ile różnych sposobów można stworzyć kod interfejsu — „czysty" język C, biblioteki MFC, WFC, procedury graficzne Visual Basica, obecnie klasy .NET i z pewnością wiele innych, których nie sposób tutaj wymienić.

Wiele z tych technologii opiera się na abstrakcjach technologii niższego poziomu. W jakim celu? Zwykle po to, aby ułatwić korzystanie z funkcji oferowanych przez warstwę niższego

poziomu, a także po to, aby umożliwić dokonywanie modyfikacji tej warstwy bez konieczności wprowadzania zmian w programach, które z niej korzystają.

Z drugiej strony, jeśli modyfikacjom poddawana jest stale warstwa abstrakcji, zaprzeczamy w ten sposób jej idei. Przyjdzie co prawda taki moment, kiedy warstwa niższego poziomu (tutaj: Windows) będzie wymagała gruntownej przeróbki przestarzałego kodu, musimy jednak przyznać, że warstwa abstrakcji zmienia się znacznie częściej.

Swego czasu autor książki użył „czystego" C do stworzenia interfejsu GDI dla 16-bitowych kontrolek VBX. Nadeszła era obiektów OLE i powstała presja, aby zmodyfikować kod GDI (i nie tylko GDI) tak, aby korzystał z biblioteki MFC. W pewnym zakresie zmiany te zostały dokonane. Po jakimś czasie pojawiła się biblioteka ATL i kolejna potrzeba modyfikacji kodu.

W większości przypadków powrócono wówczas do 16-bitowego kodu w C i poddano go konwersji. Kod ten był prostszy i łatwiejszy w zastosowaniu niż kod wykorzystujący MFC. W tej postaci prawdopodobnie już pozostanie, gdyż nie przewiduje się jego kolejnej modyfikacji w związku z pojawieniem się nowej technologii (.NET).

Uwaga

Termin .NET odnosi się do najnowszego środowiska programistycznego firmy Microsoft. Programy stworzone za pomocą tego systemu uruchamiane są we wspólnym środowisku uruchomieniowym CLR (*Common Language Runtime*). Wszystkie funkcje udostępniane przez platformę systemową zostały zawarte w bibliotekach klas utworzonych w dostępnych językach (zwykle jest to C#, VB.NET lub C++), które wykorzystują CLR.

Oczywiście postępując w ten sposób, traci się możliwość uruchomienia kodu na wielu platformach. Co może istotniejsze, rezygnuje się również z zalet, jakie zapewnia mechanizm kodu zarządzanego (*managed code*) w technologii .NET. Należy więc rozważyć, czy korzyści, jakie dałoby zastosowanie najnowszych technologii, zrekompensują koszt modyfikacji kodu.

Dzięki środowisku CLR programiści mają z pewnością więcej możliwości programowania wieloplatformowego, lecz nie należy chyba oczekiwać boomu wieloplatformowych aplikacji. Programiści będą prawdopodobnie nadal tworzyć kod specyficzny dla konkretnej platformy uruchomieniowej, aby móc w pełni wykorzystać jej możliwości. Tak więc wspomniane wyżej fragmenty kodu w C, składające się na kod źródłowy 32-bitowych komponentów, pozostaną chyba bezpieczne.

Dzięki Internetowi nastąpił jednak zwrot w kwestii wieloplatformowości. Większość kodu aplikacji internetowych wykonywana jest bowiem przez przeglądarki, które zapewniają (lub przynajmniej powinny zapewniać) niezależność od platformy systemowej. Część kodu znajdująca się natomiast po stronie serwera (odpowiedzialna za dostarczanie interfejsu aplikacji do przeglądarek użytkowników) nie musi być przecież niezależna od platformy. Uruchamiana jest bowiem na wydzielonym serwerze pod kontrolą systemu NT, UNIX lub jego pochodnych. Aplikacja stworzona dla konkretnej platformy systemowej generuje uniwersalny kod HTML (w idealnym przypadku).

W praktyce, oczywiście, rzadko ograniczamy się do zwykłego języka HTML. W celu uzyskania pożądanego efektu sięgamy po język DHTML, skrypty czy też wykorzystujemy obiekty sterujące po stronie klienta. W wielu przypadkach ograniczamy w ten sposób liczbę typów przeglądarek, które potrafią obsłużyć naszą witrynę i zaoferować użytkownikowi wszystkie jej funkcje i możliwości. Łatwo się w tej sieci zaplątać.

Znajdujemy się na etapie ewolucji i równocześnie w trakcie rewolucji. Dynamicznie rozwijają się technologie internetowe, a jednocześnie wciąż jest dużo do zrobienia w WIN32. Tam bowiem są korzenie większości z nas i z pewnością technologia ta nie straci szybko na znaczeniu.

Historia Visual Basica i C++

Historia tych dwóch języków sięga zamierzchłych czasów; zamierzchłych w odniesieniu do stosunkowo krótkiej historii komputerów. Początków VB, jak i C++, upatrywać należy w środowisku DOS (doświadczenie autora książki związane z programowaniem w systemie DOS dotyczy głównie języka Basic; w jego dorobku jest niewiele programów stworzonych w C i przeznaczonych dla tego systemu).

Linia graniczna między tymi językami została wytyczona właśnie w epoce DOS-a. Język C uchodził wówczas za narzędzie profesjonalistów, natomiast Basic traktowano jak zabawkę dla dzieci. Opinię tę Basic „zawdzięcza" w znacznej mierze jednej z jego interpretowanych wersji, GWBasic, która w owych czasach zyskała znaczną popularność. Przezywano go *Gee Wiz*[1] Basic, co charakteryzuje go całkiem dobrze. Od tamtego czasu Basic toczy mozolną walkę o zdobycie uznania w oczach programistów.

Po pojawieniu się systemu Windows programiści C znowu mogli się czuć jak u siebie w domu. Do tego systemu programiści Basica nie mieli bowiem dostępu. Do czasu, gdy na arenę wkroczył Visual Basic.

Środowisko 16-bitowe

Naszą historię musimy rozpocząć od systemów 16-bitowych, ponieważ początki Visual Basica związane są właśnie z takim środowiskiem. Trzy pierwsze edycje VB przeznaczone były dla wersji 16-bitowych. Dopiero Visual Basic 4 był wersją 32-bitową.

Grom, czyli Visual Basic 1

Thunder[2] (kryptonim projektu Visual Basic 1) oznaczał duży krok naprzód. Padały wówczas komentarze, że to tylko chwilowa moda, zabawka i tym podobne. Okazało się, że narzędzie to nie było ani przejściową modą, ani tym bardziej błahostką. Autor książki pracował wówczas w firmie MicroHelp, która stworzyła kilka z pierwszych bibliotek dla Visual Basica. To była wówczas absolutna nowość; wiele osób myślało, że to właśnie my scaliliśmy narzędzia w pasek narzędziowy!

Pierwsza wersja VB miała jeszcze wiele ograniczeń — kompilacja wyłącznie do pseudokodu, ograniczony zestaw narzędzi, zapis formularzy tylko w formacie binarnym, brak możliwości rozbudowy o dodatkowe moduły czy funkcje. Lecz bardzo ważny pierwszy krok został wykonany.

---

[1] Zwrot *gee wiz* (właśc. *gee whiz*) w potocznej angielszczyźnie (USA) wyraża zaskoczenie lub entuzjazm, w wolnym tłumaczeniu odpowiadają mu polskie: *ojej!, rety! — przyp. tłum.*

[2] Z angielskiego: grzmot, grom — *przyp. tłum.*

Narodziła się bowiem koncepcja niestandardowych kontrolek *(custom controls)*. Były to początki programowania opartego na komponentach, w formie jaką znamy obecnie.

W pierwszym zestawie firmy MicroHelp znalazło się kilka ciekawych i popularnych kontrolek. Jednym z nich było pole listy z możliwością wielokrotnego wyboru, podczas gdy pierwsze pole listy Visual Basica pozwalało na zaznaczenie tylko jednego elementu. Zestaw suwaków cechował się ciągłą reakcją, natomiast oryginalne suwaki Visual Basica aktualizowały swoją wartość dopiero po zwolnieniu przycisku myszy. Jak te czasy się zmieniają...

Idea programowania opartego na komponentach zapoczątkowała rozwój największego rynku „dodatków" do kompilatorów, jaka kiedykolwiek istniała.

Oprócz komponentów, w Visual Basicu 1 programiści Basica zetknęli się po raz pierwszy z ideą programowania przez zdarzenia (ang. event-driven programming). Do tej pory nie mieli raczej takiej okazji, chyba że ktoś zajmował się konstrukcją oprogramowania języka C/SDK.

Dla programistów przechodzących z systemu DOS do Windows programowanie przez zdarzenia było zupełną nowością. I właśnie dzięki VB liczba „przesiadających" się na system Windows była pokaźna. Ni stąd, ni zowąd programiści, którzy do tej pory nie marzyli nawet o programowaniu w Windows, otrzymali taką szansę i wielu z niej skorzystało. To dopiero rewolucja.

Pierwszymi komponentami były obiekty VBX. Mimo że w porównaniu do dzisiejszych komponentów były nieco prymitywne, miały wiele zalet. Ich tworzenie przebiegało stosunkowo prosto. Posiadały dane egzemplarza oraz dane współdzielone. Były pierwszymi elementami programowania, dla których można było definiować niestandardowe właściwości i zdarzenia. Obiekty VBX miały postać 16-bitowej biblioteki DLL wraz ze wszystkimi jej zaletami i wadami.

W środowisku 32-bitowym biblioteki DLL umieszczane są w przestrzeni pamięci programu wykonywalnego. Biblioteki 16-bitowe ładowane były natomiast do pamięci tylko przy pierwszym wywołaniu. Co prawda zachodziły przy tym komplikacje z segmentem danych oraz stosem, lecz właściwość ta miała również swoje zalety. Dzięki niej 16-bitowe biblioteki DLL mogły służyć do realizacji szybkiej komunikacji międzyprocesowej. Można było bowiem wykorzystać fakt, że w pamięci obecny jest tylko jeden egzemplarz biblioteki i wspólna dla programów przestrzeń danych, i odpowiednio dostosować swój kod.

Okazuje się, że to właśnie 16-bitowe biblioteki DLL były najbliższe pierwotnej koncepcji biblioteki dołączanej dynamicznie. Przewodnią ideą, jaka przyświecała jej twórcom, była oszczędność miejsca, zarówno w pamięci, jak i na dysku. Z biegiem czasu mieliśmy do czynienia ze stałym wzrostem poziomu złożoności obsługi bibliotek DLL (niezależnie od tego, czy nazywano je obiektami VBX, OCX, COM czy jeszcze inaczej). Obecnie inicjatywa firmy Microsoft w postaci technologii .NET zdaje się być zwrotem ku czasom, kiedy biblioteki nie były rejestrowane, lecz umieszczane razem z plikami EXE.

Pod hasłem rozwiązywania problemu określanego mianem „DLL Hell" („Piekło DLL") wprowadzano coraz to nowe innowacje, które pogłębiały złożoność mechanizmu DLL, a których większość była zupełnie niepotrzebna. Aby rozwiązać ten problem, należało przede wszystkim instalować prywatne biblioteki DLL we wspólnym katalogu z programem, który z nich korzysta. W przypadku środowiska 16-bitowego nie rozwiązało to jednak problemu „piekła DLL", gdyż tylko jedna kopia biblioteki DLL mogła znajdować się w danej chwili w pamięci. W efekcie mieliśmy do czynienia z częstym zjawiskiem zawieszania się programów w obecności innych działających aplikacji.

Rozważmy następujący przykład: 16 bitowy program ładuje bibliotekę DLL wymaganą przez inną aplikację. Może to być wcześniejsza wersja biblioteki, która nie jest zgodna z tym drugim programem. Jeśli teraz uruchomiony zostanie program korzystający z nowszej wersji biblioteki, w pamięci obecna będzie jej starsza, niewłaściwa wersja załadowana przez pierwszą aplikację. Problem ten nie dotyczy programów 32-bitowych, ponieważ w ich przypadku biblioteki DLL umieszczane są w przestrzeni kodu danego programu, mogą więc ze sobą współistnieć. Warunek jest tylko jeden — muszą zostać zapisanie na dysku w sposób gwarantujący załadowanie odpowiedniej wersji biblioteki.

W środowisku 32-bitowym umieszczenie biblioteki wraz z plikiem EXE dałoby więc pozytywny rezultat. Różne wersje bibliotek DLL mogą bowiem zostać umieszczone w pamięci i nie „przeszkadzać" sobie nawzajem. Wprowadzenie rejestru wraz z całą jego złożonością i kłopotami stanowiło więc rozwiązanie problemu, który już nie istniał.

Lecz zanim „przesiądziemy" się na platformę 32-bitową, musimy jeszcze poznać kilka faktów związanych z platformą 16-bitową.

Visual Basic 2

Ulepszenia dokonane w drugiej wersji VB nie były co prawda znaczne, lecz wprowadzona została między innymi wyjątkowa możliwość „łamania" obiektów sterujących. Opisany wyżej problem „piekła DLL" coraz bardziej dawał się programistom we znaki. Prawdopodobnie to właśnie wtedy w firmie Microsoft zdecydowano, że wszystkie informacje należy przenieść do rejestru, a na dysku pozostanie tylko jedna kopia każdej biblioteki DLL umieszczona w katalogu systemowym. Sprawy przybrały zły obrót.

Jako zaletę Visual Basica 2 można wskazać możliwość użycia kolorów w trybie *high-color* w obrębie kontrolek. Wówczas określano to mianem obsługi trybu 256-kolorowego, lecz najważniejszy był sposób obsługi palety dla kontrolki zawierającej obraz. Oczywiście, zapewniała ona operacje na większej liczbie kolorów niż 256. Nadal można było jednak korzystać wyłącznie z bitmap, ikon oraz metaplików. Sytuacja ta nieprędko miała się zmienić.

Inną nowością VB 2 były kontrolki w wersji „light" (*light controls*). Wszystkie komunikaty przeznaczone dla tych kontrolek docierały do nich za pośrednictwem formularza lub innego obiektu, na którym owa „lekka" kontrolka został umieszczony. Ponieważ kontrolki te nie posiadały uchwytu okna, zarządzanie nimi było nieco skomplikowane.

Wraz z językiem Visual Basic 2 pojawiła się funkcja VBFormat, dostępna z poziomu kodu C w obrębie obiektu VBX. Umożliwiła ona formatowanie łańcuchów za pomocą procedur Visual Basica. Cecha ta przydawała się w przypadku wielu kontrolek, lecz jej działanie było ograniczone wyłącznie do Visual Basica.

Powróćmy teraz do zmian, które dały możliwość „łamania" kontrolek. W przypadku gdy nasza kontrolka korzystała z funkcji Visual Basica 2, „musiała" sprawdzać wersję VB i ograniczać w jakiś sposób swoje możliwości podczas pracy w Visual Basicu 1. Wydaje się to całkiem oczywiste, lecz w zależności od funkcji danego obiektu nie zawsze takim było.

Ponadto niektóre właściwości kontrolki mogą być związane z funkcjami dostępnymi wyłącznie w VB 2. Problem pojawić się może kilka miesięcy po utworzeniu kontrolki, gdy zechcemy dodać do niego jakąś właściwość, która działać powinna w każdej wersji VB. Na drodze stają wówczas właściwości obsługiwane wyłącznie przez Visual Basic 2. W VB 1 należy zastąpić te właściwości ich atrapami, aby umożliwić działanie kontrolki.

Visual C++ 1

Kontrolki dla Visual Basica 1 oraz 2 były tworzone za pomocą DOS-owego edytora i DOS-owego kompilatora C++ (przynajmniej tak postępował autor niniejszej książki). Kompilowane były z linii poleceń i testowane przy użyciu programu CodeView. Wszystko uległo zmianie wraz z pojawieniem się Visual C++ 1.5.

Warto przede wszystkim wspomnieć o Visual C++1.52, który obsługiwał (chociaż obsługa ta była bardzo skromna) niestandardowe kontrolki, ponieważ ma to związek z naszą dyskusją na temat programowania komponentów. Trzeba również wymienić tę wersję tylko z tego względu, że dzięki niej została zachowana do dziś pierwotna specyfikacja niestandardowych kontrolek Visual Basica w wersji 1.0. W składzie aktualnego pakietu dystrybucyjnego MSDN język C++1.52 występuje jako obowiązująca realizacja 16-bitowego C++.

Z biegiem lat programiści powoli zapominali o tym, że pewne procedury nie były zgodne z pierwszą wersją Visual Basica. Niby nie ma się czym przejmować, gdyż Visual Basic 1.0 dawno już przeszedł do historii, prawda? Niezupełnie! W chwili pojawienia się Visual Basica 2.0 wersja 1.0 była nadal bardzo rozpowszechniona. Mniej więcej w tym samym czasie Microsoft podjął decyzję o rozszerzeniu języka C++ o obsługę standardu VBX. Zadecydowano, że obsługa ta będzie zgodna ze specyfikacją Visual Basica 1.0. Kto mógł wtedy przewidzieć, że będzie to ostatnia 16-bitowa wersja Visual C++. Tym samym specyfikacja kontrolek Visual Basica 1.0 przetrwała nadal w VC++ 1.52. Tak więc pierwsza edycja Visual C++ okazała się jednocześnie jego ostatnią 16-bitową wersją.

Visual Basic 3

Wersja ta była świadectwem postępu w tej dziedzinie. Wyposażono ją bowiem w mechanizm obsługi bazy danych. W porównaniu do możliwości systemu SQL Server lub innego zaawansowanego mechanizmu bazy danych nie była to właściwie pełna obsługa, lecz mimo to wielu programistów z niej korzystało.

Innowacją w zakresie niestandardowych kontrolek było pojawienie się po raz pierwszy możliwości wiązania danych (*data binding*). Mechanizm ten był jednak stosunkowo prymitywny i nie wykorzystywano go w poważnych projektach realizowanych w języku Visual Basic 3.

Większość osób programujących w Visual Basic 3 nadal korzystała ze znacznego wsparcia w postaci narzędzi dostarczanych przez niezależne firmy. Przykładowo w 16-bitowej wersji programu UnInstaller firmy MicroHelp zastosowano 13 niestandardowych kontrolek VBX oraz kilka bibliotek DLL stworzonych w języku C. W przypadku wersji 32-bitowej sytuacja była całkiem inna, ponieważ 32-bitowe środowisko programistyczne oferowało znacznie więcej standardowych narzędzi.

W opinii autora książki Visual Basic 3 jest najlepszą 16-bitową wersją tego języka. Jeśli istnieje potrzeba stworzenia kodu 16-bitowego, to należy skorzystać właśnie z tej wersji Visual Basica. Do czasu pojawienia się Windows 2000 kod stworzony za jego pomocą spisywał się nawet lepiej na wielu platformach (WIN9X/NT) niż niejeden 32-bitowy. Zawdzięczano to całkiem niezłej emulacji środowiska 16-bitowego, jaką oferował Windows NT, oraz istnieniu dość poważnych rozbieżności pomiędzy WIN9X a NT po stronie 32-bitowej, które były przyczyną problemów ze zgodnością w przypadku programów 32-bitowych.

Visual Basic 4

W zakresie programowania 16-bitowego Visual Basic 4 okazał się porażką. Dodano wprawdzie obsługę 16-bitowego standardu OLE, ale właściwie nie wiadomo, po co Microsoft zadał sobie cały ten trud. W owym czasie Microsoft opuszczał już platformę 16-bitową, więc 16-bitowy Visual Basic 4, a szczególnie jego dodatkowe narzędzia, nigdy nie zostały dokładnie przetestowane.

Tak więc czwarta wersja VB nigdy nie stała się poważnym narzędziem programistycznym dla środowiska 16-bitowego. Z powodu przerobienia kodu i zastosowania mechanizmu OLE, w wielu przypadkach okazywała się wolniejsza w porównaniu do analogicznego kodu stworzonego w Visual Basicu 3.

Visual Basic 4 w wersji 16-bitowej obsługiwał standard VBX. Obsługiwał również 16-bitowe kontrolki OLE (*OLE Controls*), czyli kontrolki OCX. Technologia ta narodziła się i nadal „umiera" wraz z VB4-16. Specyfikacja VBX nie została poddana modyfikacji w języku Visual Basic 4, więc ostatnia jej wersja dotyczy Visual Basica 3.

Z uwagi na fakt, że jest to najbardziej „aktualna" wersja 16-bitowego języka Visual Basic, obowiązującymi do dziś standardami dla środowiska 16-bitowego są: specyfikacja VBX z Visual Basica 3 oraz specyfikacja 16-bitowych obiektów OCX z VB4. Tak więc jedyną 16-bitową specyfikacją kontrolek Visual Basica, która została całkowicie wycofana, jest wersja 2 standardu VBX. Wszystkie pozostałe przetrwały do dziś — VBX 1.0 w Visual C++ 1.52, VBX 3.0 oraz 16-bitowe OCX w Visual Basicu 4.

Visual Basic 4 — pierwsze podejście

Największy nacisk w czwartej wersji Visual Basica położono na oprogramowanie działające w środowisku 32-bitowym. Oprogramowanie to sprawiało niestety sporo problemów, podobnie jak obsługa 32-bitowych obiektów OCX. Mimo to należy uszanować spory wysiłek, jaki podjęto w związku z czwartą wersją Visual Basica, która była jedyną edycją tego języka zawierającą obie jego realizacje, 16- i 32-bitową.

Nie wiadomo właściwie, dlaczego Microsoft podjął decyzję o uwzględnieniu obydwu wersji Visual Basica w jednej edycji. Było to trudne przedsięwzięcie zarówno dla samego Microsoftu, jak i dla innych firm, producentów różnych dodatków do Visual Basica. Pojawiła się bowiem konieczność tworzenia i testowania dwóch wersji każdego produktu. Ucierpiały na tym przede wszystkim wersje 16-bitowe, a i dla 32-bitowych nie była to sytuacja korzystna.

Microsoft jasno dał do zrozumienia, że ścieżka dalszego rozwoju związana będzie ze środowiskiem 32-bitowym. Dziwi zatem fakt, że firma w ogóle wzięła pod uwagę 16-bitową wersję tego kompilatora. Visual Basic 3 był 16-bitowym środowiskiem o całkiem sporych możliwościach. Tylko nieliczne zadania wymagały użycia Visual Basica 4-16, gdyż nie można ich było zrealizować w VB 3. Jedynie w takich przypadkach warto było sięgnąć po Visual Basica 4.

Jedną z takich nowości był mechanizm automatyzacji OLE (*OLE automation*). Visual Basic 3 nie posiadał wbudowanej obsługi tej technologii. Jeśli wymagane było stworzenie 16-bitowego serwera automatyzacji OLE, pomocny okazywał się VB 4. Jednym z zastosowań 16-bitowej automatyzacji mogło być stworzenie swego rodzaju *thunkera* (*thunking* to metoda umożliwiająca wywoływanie procedur z 16-bitowych bibliotek DLL z poziomu 32-bitowego kodu[3]). System nie dokonywał bowiem sprawdzenia, czy serwer automatyzacji OLE jest 16- czy 32-bitowy, nie było więc przeszkód do wykorzystania 16-bitowego serwera automatyzacji OLE z poziomu programu 32-bitowego.

Mechanizm automatyzacji OLE Visual Basica 4 dawał zatem możliwość użycia 16-bitowego kodu z poziomu 32-bitowego programu. Nie był to oczywiście najszybszy sposób wywoływania procedur, więc w miarę możliwości lepiej było stosować „prawdziwy" *thunking*. *Thunking* jest jednak silnie zależny od platformy systemowej, natomiast automatyzacja OLE — przeciwnie.

Można oczywiście podawać powody, dla których warto było użyć Visual Bascia 4-16, lecz realna konieczność pojawiała się bardzo rzadko, a sama platforma sprawiała zbyt dużo problemów. Nie warto było się zatem „przesiadać" na VB4-16 bez wyraźnego powodu. Znacznie lepszym rozwiązaniem dla programisty była bezpośrednia „migracja" do świata 32 bitów.

Środowisko 32-bitowe

Środowisko programowania WIN32 jest dziś „miejscem pracy" dla większości z nas. Pozostało niewiele zadań realizowanych z wykorzystaniem 16-bitowych technologii języka Visual Basic.

Visual Basic 4

Jak już wcześniej wspomniano, w czwartej wersji Visual Basica największy nacisk położono na oprogramowanie działające w środowisku 32-bitowym. Ten kompilator nie był niczym niezwykłym w porównaniu do postępów dokonanych w wersji 3.

Visual Basic 4 był pierwszą próbą przeniesienia tego języka na platformę 32-bitową. Kompilator ten oferował ponadto niewiele funkcji, aby warto było z niego korzystać. Nadal był jedynie kompilatorem pseudokodu.

Visual Basic 5 — Visual Studio 5

W postaci piątej wersji Visual Basica programiści otrzymali bardzo solidne narzędzie. Możliwość kompilacji do kodu macierzystego oraz spory zestaw narzędzi uzupełniających, dzięki

[3] I odwrotnie — procedur 32-bitowych z poziomu 16-bitowego kodu — *przyp. tłum.*

któremu w większości przypadków dodatkowe zakupy nie były konieczne, składały się na kompletny system programowania. Po raz pierwszy w języku Visual Basic pojawia się możliwość tworzenia kontrolek ActiveX.

Trudno zrozumieć, dlaczego możliwość tworzenia formantów ActiveX w Visual Basicu nigdy nie była wykorzystywana na szerszą skalę. To przecież wspaniały sposób na hermetyzację kodu, zarówno w przypadku pracy nad komercyjnym projektem, jak i przy projektach wewnętrznych. Od momentu pojawienia się Visual Basica 5 tworzenie niestandardowych formantów stanowiło nieodłączny element prac we wszystkich istotnych projektach realizowanych przez autora książki. Być może dopiero technologia .NET sprawi, że tworzenie formantów stanie się popularną praktyką.

Wraz z wprowadzeniem języka Visual Basic 5 firma Microsoft umieściła kilka produktów programistycznych w jednym pakiecie. Pakiet Visual Studio 5 był pierwszym systemem oferującym kompletne rozwiązanie, włącznie z kompilatorem C++. W ten sposób kompilator C++ trafił „pod strzechy". Pierwszym 32-bitowym kompilatorem C++ był Visual C++ 2.0, ale dopiero wersja 4.0 była częściej używana przez autora książki. Z numeracji kolejnych wersji języka — od 2.0 do 5.0 — wynika, że w środowisku 32-bitowym borykano się z pewnymi początkowymi kłopotami. Wersja oznaczona numerem 5 była pierwszą, która weszła w skład nowego zestawu Visual Studio.

Skoro więc podarowano nam wspaniałe narzędzie w postaci C++, to dlaczego go nie wykorzystać? Rodziły się przeróżne pomysły stosowania kompilatora lub środowiska programistycznego Visual C++ 5 — od tych prostych po bardziej zaawansowaane.

Jednym z „delikatnych" sposobów wykorzystania środowiska Visual C++ było użycie takich funkcji, jak *Find in Files* (wyszukiwanie słów lub fraz w plikach). Dla samej tej funkcji warto było uruchomić edytor Visual C++ 5. Realizuje ona wyszukiwanie określonych łańcuchów tekstowych w plikach. Można ograniczać zakres poszukiwań do plików określonego typu. Podczas wyszukiwania przeglądane są również podkatalogi. Po jego zakończeniu wyświetlana jest lista wierszy z plików, które odpowiadają wzorcowi. Dwukrotne kliknięcie pozycji na liście powoduje załadowanie pliku do edytora i ustawienie kursora w miejscu wystąpienia szukanego tekstu.

Możliwość rejestracji makropoleceń w edytorze do celów zaawansowanej edycji występuje dopiero w najnowszej wersji Visual Basica. W Visual C++ można nawet uruchamiać i testować programy napisane za pomocą Visual Basica, jeśli tylko zostały skompilowane z odpowiednimi ustawieniami.

Bardziej zaawansowane metody wykorzystania C oraz C++ są tematem tej książki. Visual Studio 5 jest pierwszym systemem, dzięki któremu wszystkie przedstawione tu pomysły można realizować w praktyce, bowiem wraz z tą wersją programiści po raz pierwszy otrzymali wszelkie niezbędne kompilatory. Oczywiście większość programistów Visual Basica nadal nie używa samego kompilatora C++.

Visual Studio 6

W Visual Basicu 6 wzbogacono możliwości dostępu do danych. Dodano również nowe kontrolki oraz narzędzia w formie kreatorów. Jednakże środowisko programistyczne nie uległo zasadniczej zmianie w porównaniu do jego piątej wersji.

W Visual Basicu 6 pojawiła się funkcja raportowania. Trzeba przyznać, że nie była to rewelacja, o czym świadczy chociażby fakt, że sam Microsoft w technologii .NET powrócił do narzędzi firmy Crystal[4]. Całkiem udaną nowością było natomiast wprowadzenie struktury hierarchicznej do zestawu rekordów oraz komponentu FlexGrid. Hierarchiczne zestawy rekordów zdobyły wielu zwolenników.

Inną przydatną innowacją była możliwość tworzenia źródeł danych. Rozłączone zestawy rekordów oraz możliwość przekazywania zestawu rekordów w wywołaniu procedury, nawet pomiędzy procesami, oznaczały wyraźny postęp. Umożliwiało to prostą konstrukcję wielowarstwowego kodu (*multi-tier code*).

Z punktu widzenia autora tej książki najbardziej pożyteczną zmianą w Visual Studio 6 było pojawienie się trzeciej wersji biblioteki ATL. Od tej pory tworzenie kontrolek w tej technologii stało się praktyczniejsze i łatwiejsze w porównaniu do wcześniejszych wersji ATL.

Można by jeszcze wymienić kilka zmian, jakie dokonane zostały w Visual Studio 6, lecz żadna z nich z pewnością nie okazała się rewelacją. Po rekompilacji projektu stworzonego w Visual Basicu 5 nie czekało nas nic szczególnego. Ot, zwykła codzienność.

.NET

Najnowsza technologia pochodząca z firmy Microsoft burzy dotychczasowe reguły. Jedynie język Visual C/C++ funkcjonuje zasadniczo tak jak dawniej. Nie ma już ani Visual J++, ani Visual Interdev, a Visual Basic został oparty na modelu klas (tak jak wszystkie języki .NET). To nie jest już stary, poczciwy Visual Basic, znany nam do tej pory. Wymagany jest poziom zaawansowania, jakiego VB nigdy dotąd nie wymagał. Konieczne jest również dostosowanie lub wręcz ponowne napisanie większości fragmentów kodu w celu uruchomienia ich na nowej platformie.

Gdy Microsoft poinformował, że Visual Basic korzystać będzie z mechanizmu dziedziczenia implementacji, wydawało się, że będzie to opcjonalna możliwość (podobnie jak w Visual C++). Można przecież programować zarówno w C, jak i w C++. Nic podobnego. Języki .NET oparte zostały na klasach i jest to jedyna akceptowana metoda programowania.

Podobnie jest z językiem C#. „Migracja" ze środowiska Visual Basic 6 do Visual Basic .NET może być równie trudna, co bezpośrednie przejście na język C#. Wielce prawdopodobne jest, że wielu programistów właśnie tak postąpi — „przesiądzie" się wprost na język C#.

Jeśli podnosimy kwestię programowania w językach, które korzystają ze wspólnego środowiska uruchomieniowego CLR (*Common Runtime Language*), to poruszamy zagadnienie tworzenia kodu zarządzanego oraz kodu niezarządzanego. Kod zarządzany (*managed code*) działa pod kontrolą środowiska CLR; nosi taką nazwę, ponieważ środowisko zwalnia programistę z obowiązku „odśmiecania" (ang. garbage) pamięci z niepotrzebnych obiektów i wykonuje za niego całą niewdzięczną pracę. Kod niezarządzany natomiast to klasyczny kod wykonywany poza

[4] Przypis: Crystal Reports jest obecnie najczęściej używanym narzędziem do tworzenia raportów, wiecej można znaleźć na stronie *http://www.crystaldecisions.com/*

środowiskiem CLR. W językach C oraz C++ tworzyć można kod niezarządzany (chociaż dostępne są rozszerzenia do C++ umożliwiające tworzenie kodu dla CLR), natomiast w Visual Basicu oraz w C#, które korzystają z nowej technologii, tworzony jest kod zarządzany.

Znaczenie języka Visual Basic

Język Visual Basic zrewolucjonizował metody tworzenia programów dla systemu Windows. Łatwość konstruowania interfejsu użytkownika oraz prostota programowania nie mają sobie równych wśród dostępnych narzędzi, być może z wyjątkiem tworzenia kodu zarządzanego w technologii .NET. Visual Basic jest niewątpliwie wspaniałym środowiskiem, mimo to nie należy przeceniać jego możliwości. Wielu programistów popełnia błąd, próbując w tym języku zrealizować zadania, które nie są zgodne z jego przeznaczeniem. Najczęściej efekt jest taki, że powstaje kod skomplikowany, trudny zarówno w modyfikacji, jak i w testowaniu, a ponadto zwykle występują również problemy z jego stabilnością.

Użycie właściwego narzędzia przynosi zwykle efekt w postaci lepszego rezultatu końcowego. Są zadania, w których dobór narzędzia nie jest aż tak istotny, w przypadku innych zaś to kwestia kluczowa. Nie należy unikać zastosowania języka C czy C++, jeśli sytuacja tego wymaga. Celem tej książki jest właśnie zapoznanie i „oswojenie" czytelnika z tymi narzędziami.

Mocne strony Visual Basica

Mechanizm kodu zarządzanego w systemie .NET wyrównał możliwości wielu języków. W celu porównania cech poszczególnych narzędzi należy łącznie potraktować większość języków opartych na kodzie zarządzanym. Pod względem oferowanych możliwości język C#, na przykład, jest bliższy językowi Visual Basic niż C++. Środowisko projektowe .NET wywodzi się z poprzednich wersji Visual Basica.

Bezapelacyjnie najlepsze narzędzie do projektowania interfejsu

Visual Basic jest prawdopodobnie najpotężniejszym narzędziem programistycznym typu RAD dla WIN32. Programiści zawsze chętnie po nie sięgali z tej prostej przyczyny, że stworzenie w nim interfejsu zajmuje zaledwie kilka minut. Dzięki technologii .NET możliwości te stały się dostępne również dla wielu innych języków, lecz C oraz C++ pozostały nadal nie zmienione.

Ścisła kontrola typów

W języku Visual Basic od zawsze była stosowana ścisła kontrola typów zmiennych (*strong typing*). Unika się w ten sposób potencjalnych błędów wynikających ze stosowania wskaźników oraz kłopotów związanych z rzutowaniem typów. Oba te mechanizmy nie są bowiem dostępne w Visual Basicu.

W Visual Basicu możliwa jest natomiast konwersja typów. Automatyczna konwersja typu zmiennej przeprowadzana jest wszędzie tam, gdzie kompilator „uzna" to za właściwe. Ponieważ mechanizm taki stanowi potencjalne źródło niezamierzonych błędów, lepiej jest dokonywać jawnej konwersji zmiennych. Nie należy mylić zmiennych typu Variant oraz mechanizmu konwersji typów z tzw. luźną kontrolą typów (*loose typing*), ponieważ jedno z drugim nie ma nic wspólnego. Zarówno dla zmiennych typu Variant, jak i dla konwersji typów zdefiniowany jest ścisły zbiór zasad dotyczących konwersji, natomiast w przypadku luźnej kontroli typów takie reguły nie istnieją.

Interaktywne środowisko uruchomieniowe

Interaktywne środowisko w połączeniu z bogatymi możliwościami w zakresie projektowania interfejsu — to właśnie Visual Basic, jaki znamy. Wielu programistów wybiera to narzędzie ze względu na możliwość tworzenia, uruchamiania, testowania oraz modyfikowania kodu „na gorąco", bez konieczności opuszczania środowiska. Pod tym względem VB nie ma sobie równych.

Wsparcie ze strony producentów narzędzi

Programiści Visual Basica zawsze mogli liczyć na silne wsparcie ze strony niezależnych firm. Liczba tych dostawców wzrosła z 8 – 10 na samym początku do setek w chwili obecnej. Oferta wszelkiego rodzaju dodatków przeznaczonych dla Visual Basica jest niezwykle bogata, a trend ten utrzyma się również w przypadku platformy .NET.

Wszystko to znacznie ułatwia i usprawnia pracę programisty VB. Otrzymuje bowiem starannie napisane i przetestowane narzędzia, stworzone w języku C przez specjalistów z danej dziedziny.

Słabości języka Visual Basic

Mimo że Visual Basic jest wspaniałym narzędziem typu RAD, właśnie to ukierunkowanie czyni go nieodpowiednim dla niektórych zadań. Programiści płacą pewną cenę za pomoc języka VB. Z jednej strony mamy bowiem łatwiejszą i wydajniejszą pracę, z drugiej zaś — utrudniony dostęp do warstwy systemowej. Wskazanie słabych punktów Visual Basica i nauczenie czytelnika, w jaki sposób ma wypełniać te luki poprzez zastosowanie języków C/C++, stanowi cel tej książki.

Szybkość i wydajność

Stale podejmuje się temat szybkości kodu Basica. Język ten nigdy nie pozbył się etykiety zabawki, którą przypięto mu z powodu interpretowanego Basica, działającego na oryginalnym komputerze PC pod kontrolą systemu DOS. Pierwsze wersje VB rzeczywiście potwierdzały opinię o małej szybkości Basica. Swoje wolniejsze działanie Visual Basic zawdzięczał tzw. pseudokodowi (*P-code*), na który tłumaczone były programy (zamiast tego powinny zostać poddane kompilacji do postaci wykonywalnej).

Począwszy od wersji 5 mamy już możliwość pełnej kompilacji programów. W wersjach 5 oraz 6 zastosowano właściwie ten sam kompilator co w Visual C. Szybkość wykonywania była więc nareszcie porównywalna. Pozostały jedynie różnice wynikające z realizacji pewnych funkcji w bibliotece uruchomieniowej Visual Basica. Niektóre z tych funkcji były nadal dosyć powolne, na przykład operacje na łańcuchach tekstowych.

Jednym z najbardziej niewydajnych rozwiązań, które pojawiło się po raz pierwszy w czwartej wersji VB, było zastosowanie łańcuchów w formacie Unicode. Od tej pory w Visual Basicu operacje wykonywane są bowiem wewnętrznie na łańcuchach Unicode, lecz ze względu na konieczność współpracy z systemami opartymi na zestawie znaków ANSI Microsoft postanowił konwertować łańcuchy na zestaw ANSI przy wszystkich odwołaniach do API. Odwrotna konwersja wykonywana jest oczywiście podczas powrotu z danej funkcji API.

Wraz z technologią .NET Microsoft powraca do koncepcji tzw. kodu pośredniego IL (*Intermediary Language*). Tym razem firma Microsoft dołożyła starań, aby kompilator JIT (*Just In Time*) języka IL działał wydajnie. Jeśli istnieje jednak potrzeba kompilacji do kodu maszynowego, to taką operacje można zlecić kompilatorowi JIT podczas instalacji projektu.

Ścisła kontrola typów

Jak już wspomniano, w języku Visual Basic od zawsze była stosowana ścisła kontrola typów zmiennych, która wraz z brakiem wskaźników sprawia, że trudno jest w tym języku znaleźć eleganckie rozwiązanie pewnego problemu. Polega on na tym, że typ danych nie jest znany do czasu uruchomienia programu. Klasycznym przykładem takiej sytuacji jest funkcja API `SendMessage`, której parametr `lp` może być zmienną niemalże każdego typu, tj. wskaźnikiem do danych dowolnego typu (włącznie ze strukturami i danymi łańcuchowymi), wartością typu `int` lub `long` bezpośrednio lub kombinacją wartości, np. wartościami typu `twp`.

Ten typ danych występuje dość często w systemach komunikatów, w których muszą być obsługiwane dane różnego typu. A zatem mechanizm ścisłej kontroli typów ma również ujemne strony.

W języku Visual Basic dostępny jest co prawda typ Variant, nie narusza on jednak zasady ścisłej kontroli typów. Wewnętrznie zmienna typu Variant może zawierać dane różnego typu, lecz na zewnątrz jest tylko zmienną typu Variant i wszystkie odwołania do niej muszą być również typu Variant. Zmienna typu Variant nie „zrozumie" ani łańcucha języka C, ani tym bardziej typu zdefiniowanego przez użytkownika. Obydwa typy są często stosowane wraz z omawianą wyżej funkcją `SendMessage`.

Przechowywanie wartości właściwości obiektów poza kodem programu

Projektowanie interfejsu to w Visual Basicu prawdziwa przyjemność. Można mieć jednak zastrzeżenia co do sposobu przechowywania wartości właściwości kontrolek oraz odnośnie do ograniczenia możliwości modyfikowania właściwości wielu obiektów do etapu projektowania.

Wartości właściwości obiektów są zapisywane do pliku formularza (*.frm*), a dane binarne do pliku *.frx*. Nie sposób zwykle określić, kiedy właściwości zostają zmodyfikowane i różnią się od wartości domyślnych. Utrudnia to testowanie programu w przypadku obiektów, których właściwości były zmieniane podczas projektowania. A takie postępowanie jest nieuniknione, ponieważ

w języku Visual Basic jest wiele kontrolek o właściwościach modyfikowalnych jedynie na etapie projektowania. Skutecznie uniemożliwia to zastosowanie pewnego rodzaju obejścia, polegającego na umieszczeniu zmian właściwości w kodzie podczas ładowania formularza.

Jeśli ustawienia właściwości znajdują się wewnątrz kodu programu, mamy możliwość śledzenia zmian właściwości. W środowisku .NET dokonano zmian na lepsze. Nie ma już mechanizmu przechowywania właściwości, który powodowałby zapisywanie ich w jakimś specjalnym miejscu. Ustawienia właściwości umieszczane są bezpośrednio w kodzie programu, i tak jest chyba najlepiej.

Znaczenie języka C++

Język C++ zawsze lepiej sprawdzał się w zadaniach, które wymagały bliższego kontaktu z systemem. Oczywiście przy jego użyciu wielokrotnie tworzono również interfejs użytkownika, jednak ten obszar zastosowań nie jest mocną stroną C++.

Analizując tradycyjne zastosowania C++ w porównaniu do Visual Basica, można zauważyć istotne różnice. Kod użytkowy powstawał zwykle w językach C oraz C++, natomiast za pomocą VB realizowano projekty typu RAD. Nawiasem mówiąc, sam Visual Basic został napisany w C++. Ponadto w języku tym powstaje większość kontrolek dla VB, dostarczanych przez niezależne firmy. Większość aplikacji wchodzących w skład pakietu Microsoft Office również stworzono w języku C/C++.

Chociaż większość z nas nie uczestniczy w procesie tworzenia kodu użytkowego na tym poziomie, to zdajemy sobie sprawę, że musiały istnieć powody takiego wyboru. Język C++ pozwala zwykle na stworzenie bardziej wydajnego kodu z tej prostej przyczyny, że oferuje więcej możliwości w zakresie obsługi pamięci oraz współpracy z systemem. Szerokie możliwości optymalizacji oraz dostosowania kodu do konkretnego zadania pozwalają na uzyskanie, w zależności od wymagań, kodu bardziej efektywnego lub o mniejszych rozmiarach. Nawet na platformie .NET można napotkać problemy, w przypadku których wybór języka C/C++ okazuje się najlepszym, a czasem jedynym rozwiązaniem.

Mocne strony C++

Visual C++ istotnie posiada pewne cechy, które czynią go najlepszym narzędziem do realizacji pewnych zadań programistycznych. Pojawienie się technologii .NET zniwelowało w pewnym stopniu przewagę tego języka, jednak nadal pozostały obszary zastosowań, które są domenami C/C++. Poniżej wyszczególnimy główne zalety języka i omówimy ich związek z technologią .NET.

Dziedziczenie implementacji

Mechanizm dziedziczenia implementacji jest popularną strategią programistyczną, pozwalającą na korzystanie ze wspólnych metod zaimplementowanych w klasie bazowej przez podklasy,

które się z niej wywodzą. Dzięki dziedziczeniu znacznie prościej jest ponownie wykorzystać własną pracę.

Dziedziczenie implementacji zawsze dawało przewagę Visual C nad Visual Basiciem. Przełom nastąpił wraz z wprowadzeniem technologii .NET, dzięki której programiści Visual Basica po raz pierwszy w historii otrzymali możliwość korzystania z dobrodziejstw tego mechanizmu.

Słabsza kontrola typów, możliwość rzutowania typów

Słabsza kontrola typów w C oraz C++ pozwala na rzutowanie typów. W językach tych istnieje również pojęcie wskaźników do zmiennych. Te dwie cechy czynią język C/C++ bardzo elastycznym, jeśli chodzi o sposób operowania danymi przez kompilator. Jeśli na przykład istnieje potrzeba, aby wartość typu `long` stała się nagle wskaźnikiem do zmiennej łańcuchowej, to nie ma nic prostszego — wystarczy wykonać operację rzutowania typów i już mamy łańcuch. Należy przy tym oczywiście upewnić się, czy dana wartość będzie prawidłowym wskaźnikiem typu łańcuchowego, jednak teraz istotne jest to, że operacja taka jest w ogóle dozwolona.

Opisane tu możliwości sprawiają, że język C idealnie nadaje się do manipulowania danymi w systemie rozpowszechniania komunikatów, jakim jest np. Windows. Nieważne, czym w rzeczywistości jest dana wartość `long` — liczy się to, że można ją bez trudu poprawnie zrzutować i że zostanie prawidłowo potraktowana przez kompilator. Realizacja tej samej operacji w języku Visual Basic wymaga karkołomnych wyczynów, polegających najczęściej na kopiowaniu bloków pamięci w miejsce zajmowane przez zmienną docelowego typu.

Możliwe jest prawie wszystko

Różnorodność operacji, które można wykonać w języku C++, czyni to narzędzie bardzo elastycznym. W C/C++ możliwe są operacje na wskaźnikach oraz bezpośrednio na pamięci. Język ten działa na bliższym, w porównaniu z VB, poziomie do API systemu Windows. Interfejs API zaprojektowany został pod kątem programów tworzonych w języku C, a więc typy danych oraz struktury idealnie pasują do programów pisanych w C czy C++.

Język C lub C++ jest również odpowiednim narzędziem do tworzenia kodu niskopoziomowego, takiego jak sterowniki oraz usługi systemowe.

Żaden inny język nie może się pochwalić taką liczbą przykładowych programów, jaka jest dostępna dla języka C/C++. Dla VB, oczywiście, również istnieje mnóstwo przykładów, jednak są obszary, dla których dostępne są liczne próbki w języku C, a które nie zostały nawet oprogramowane w języku Visual Basic.

Słabości języka C++

Te same cechy, które decydują o zaletach języków C oraz C++, mogą stanowić również ich słaby punkt. Z faktu, że język ten z natury jest językiem niższego poziomu i na tym poziomie oferuje dostęp do systemu, wynikają następujące ograniczenia:

- **Właściwie nic nie jest proste** — to jest cena, którą się płaci za elastyczność języka. W języku C/C++ programuje się znacznie trudniej w porównaniu do Visual Basica. Operacje na wskaźnikach oraz rzutowanie typów są, w najlepszym przypadku, kłopotliwe. Dzięki wykorzystaniu klas uzyskujemy lepszą technikę kodowania, lecz jest ona trudniejsza do opanowania.
- **Projektowanie interfejsu użytkownika jest, delikatnie mówiąc, męczące** — w porównaniu z VB tworzenie interfejsu użytkownika jest archaiczne. Nie ma tu bowiem narzędzi-projektantów, które towarzyszyły Visual Basicowi od samych początków. Dzięki klasom realizacja interfejsu jest wprawdzie łatwiejsza w C++ niż w „czystym" C, lecz mimo to daleko mu jeszcze do VB. W gruncie rzeczy może się okazać, że właśnie ze względu na obecność klas programowanie w VB.NET będzie dla niektórych trudniejsze niż przy użyciu tradycyjnego Visual Basica.
- **Słabsza kontrola typów** — słabsza kontrola typów ma oczywiście również swoje negatywne strony. Wskaźniki oraz rzutowanie są mechanizmami skomplikowanymi do opanowania oraz trudnymi w realizacji. Możliwości popełnienia błędów jest wiele, a pomyłki te zwykle okazują się katastrofalne dla programu.

Język C#

Język C# jest najnowszą propozycją firmy Microsoft z rodziny języków C. Ponieważ analogicznie do innych języków w technologii .NET bazuje on na wspólnym środowisku uruchomieniowym CLR, nie jest kolejną wersją C w tradycyjnym rozumieniu.

Nieco dziwnym wydaje się fakt, że Microsoft pozostawił możliwość tworzenia kodu niezarządzanego w C oraz C++, a nie przewidział takiej opcji dla Visual Basica. Wiele osób preferujących „starego" Basica może zrezygnować ze zmiany wersji Visual Studio właśnie z powodu jego braku wśród nowych języków.

W jaki sposób język C# mieści się w ramach naszego celu, jakim jest nauka programowania w C oraz C++? Jest wielce prawdopodobne, że programiści, którzy „przesiądą" się z Visual Basica na platformę .NET, wybiorą właśnie język C#. Dlaczego? Ponieważ wierność Visual Basicowi nie ma już specjalnego uzasadnienia. Składnia klas w C# jest prostsza i bardziej logiczna w porównaniu do VB.NET. Ponadto są zadania, których realizacja jest możliwa w C#, przeciwnie niż w Visual Basic .NET.

Język C# w porównaniu do VB.NET ma podobne zalety. Wyposażony będzie w ten sam moduł projektanta co Visual Basic; również środowisko uruchomieniowe jest wspólne dla obu języków. Największe różnice pomiędzy językami .NET dotyczyć będą składni.

Język C# odziedziczył po C++ wiele zalet i wad. Najistotniejszą różnicą jest fakt umieszczenia C# „na szczycie" wielkiego środowiska uruchomieniowego (CLR). Będzie to miało niewątpliwie wpływ na jego wydajność oraz na łatwość dystrybucji.

Język C# ma wielką szansę, aby stać się przyszłością narzędzi programistycznych firmy Microsoft. Będzie to podstawowy język technologii .NET, a także ten najczęściej używany wewnętrznie w firmie Microsoft.