

Початковий курс програмування Visual Basic

Зміст

Вступ.....	2
Урок № 1. Що таке Visual Basic?.....	2
Урок № 2. Що може Visual Basic?.....	4
Урок № 3. Інсталяція та налаштування Visual Basic.....	4
Урок № 4. Для тих, хто ніколи не.....	4
Урок № 5. Етапи розробки додатка.....	6
Урок № 6. Структура проекту Visual Basic.....	6
Урок № 7. Середовище розробки Visual Basic.....	7
Урок № 8. Легкість роботи з кодом Visual Basic.....	8
Урок № 9. З чого складається код?.....	11
Урок № 10. Масиви, записи і перерахування.....	18
Урок № 11. Вирази.....	23
Урок № 12. Оператори.....	25
Урок № 13. Керуючі структури.....	27
Урок № 14. Процедури і функції.....	31
Урок № 15. Зводимо все разом.....	34
Урок № 16. Налагодження програми.....	41
Урок № 17. Додводимо до розуму.....	48
Урок № 18. Компіляція.....	52
Висновки.....	56

Вступ

Отже, Ви вирішили вивчити мову програмування високого рівня - Visual Basic. Я спробую допомогти вам у цьому. Наберіться трохи терпіння, бажання і вперед, у простори Visual Basic! Адже Visual Basic - це Міцний Горішок! ;)

Даний курс призначений для тих, хто:

- ніколи не програмував, але хоче навчитися
- програмував на іншій мові (Turbo Pascal, QBASIC)
- програмував на іншій мові високого рівня (C++, Delphi)

Як бачите аудиторія досить широка. Від Вас потрібно лише невеликий досвід роботи з Windows. І все.

Даний курс, цілком зрозуміло, не претендує на повноцінний підручник по Visual Basic. Він є тільки вступом до програмування на Visual Basic. Я спробував розібрати основні моменти і прийоми програмування на Visual Basic, знаючи які, Ви без особливих зусиль зможете поповнювати Ваші знання.

Те, що не ввійшло в даний матеріал, Ви зможете прочитати в статтях, які я буду час від часу розміщувати на сайті.

Удачі! І Вперед!

Урок № 1. Що таке Visual Basic?

Перед програмістами-початківцями завжди постає одне й те саме питання, а саме, яку мову програмування вибрати? На чому програмувати? Можу сказати, що краще починати з легкої і в той же час потужної мови - Visual Basic. Вивчивши прийоми програмування на Visual Basic, ви зможете без особливих зусиль вивчити інші мови, такі як Pascal, C++ і ін.

Слово “БЕЙСИК” (BASIC) – “базовий, основний” - утворено з початкових букв англійського вислову “Універсальна мова символічного кодування для початківців”. Саме оце “для початківців” довго викликало зневагу програмістів, причому подібна зневага не зникла дотепер, незважаючи на наявність професійних видань Visual Basic.

Перший IBM PC мав 16-розрядний бейсік - BASICA, розроблений IBM, який трохи згодом був посунутий мікрософтовським GW-BASIC і QUICK-BASIC. В останньому була прибрана нумерація рядків та додано компілятор, який перетворює бейсік-програму в повноцінний exe-файл. Нарешті в 1992 році фірмою Microsoft був випущений VB 1.0 - дуже проста мова програмування для Windows 3.1. Потім були випущені VB 3.0, VB 4.0, VB 5.0 і нарешті VB 6.0. Останні дві версії мало чим відрізняються, у шостій версії поліпшено ядро та додані декілька нових функцій. Існує ще мова макросів для додатків Microsoft Office (Word, Excel і т.п.). Він називається VBA (Visual Basic for Application). За його допомогою можна маніпулювати додатками Office.

Додатки, написані на Visual Basic відрізняються від звичайних додатків тим, що вимагають для своєї роботи бібліотеку msvbvmX0.dll, яка мусить бути присутня у каталозі Windows\System. У ролі X виступає версія компілятора VB. Для VB5 - msvbvm50.dll, для VB6 - msvbvm60.dll. Ці бібліотеки йдуть у комплекті з WinME (для обох версій) і Win98 (тільки msvbvm50.dll), і природно з більш новими версіями цих операційних систем. Ви не

повинні боятися того, що разом з вашим додатком, вам доведеться тягати за собою цю бібліотеку. Вона вже є в 90% користувачів. Але якщо Ви все-таки боїтеся за аудиторію, то можете зашити бібліотеку прямо в EXE файл. Правда для цього прийдеться використовувати засоби не вхідні до складу Visual Basic. Наприклад, це можна зробити за допомогою програми Fusion (фірми BitArts).

Існує думка, що додатки Visual Basic - це не повноцінні програми, а лише псевдокод, який під час запуску виконується інтерпретатором. Це зовсім не так. Якщо ви компілюєте програму в Native Code, то отриманий EXE – це повноцінний додаток Win32, який просто використовує функції msvbvm-бібліотеки. А от P-Code є псевдокодом. Вид компіляції ви можете вказати в меню Project->Project Properties...

Середовище Visual Basic може з успіхом використовуватися користувачами-початківцями для пізнання секретів програмування і захоплюючих занять по створенню нескладних (для початку) додатків і, у той же час, надає потужні інструменти розробки досвідченим програмістам. Розвинута довідкова система дозволить при створенні додатка і роботі в Visual Basic знайти вихід з будь-якої ситуації та одержати відповідь на будь-яке питання. Починати працювати з Visual Basic можна практично з будь-яким рівнем підготовки.

Visual Basic доступний у трьох редакціях, кожна з яких забезпечує певний набір інструментів розробки. Це:

- Standard Edition

Visual Basic Standard Edition дозволяє створювати досить потужні додатки для Microsoft Windows 95 і Windows NT. Ця редакція включає всі вбудовані елементи керування Visual Basic, включаючи зв'язані (data-bound) елементи керування.

- Professional Edition

Редакція Professional забезпечує повний функціональний набір інструментальних засобів для розробки професійних рішень, призначених для тиражування. Вона включає всі можливості Standard Edition плюс додаткові елементи керування Active, включаючи елементи керування для Internet і генератор звітів Crystal Reports для Visual Basic (розглядається в главі 5).

Примітка: Елемент керування Active - це об'єкт, який можна розміщувати на формі для активації або розширення взаємодії користувача з додатком. З елементами керування Active асоціюються певні події, і вони можуть бути включені в інші елементи керування. Ці елементи керування мають розширення імені файлу .osx.

- Enterprise Edition

Редакція Enterprise дозволяє створювати розподілені додатки силами групи розробників. Вона забезпечує всі можливості редакції Professional і включає також додаткові функції, такі, як Automation Manager, Component Manager, інструментальні засоби керування базами даних і Microsoft Visual SourceSafe — проектно-орієнтована система керування версіями продуктів.

Йдемо далі. Отже, що ж може Visual Basic?

Урок № 2. Що може Visual Basic?

У принципі, можливості Visual Basic нічим не обмежені. Ви можете розширювати можливості Visual Basic за допомогою використання додаткових функцій. Visual Basic дозволяє використовувати бібліотеки динамічного компонування (DLL-бібліотеки), які також можуть сильно розширити можливості Visual Basic. Ці бібліотеки можуть бути написані на будь-якій мові програмування.

На Visual Basic можна написати будь-яку програму, - починаючи від програми, яка обслуговує рутинні операції введення даних, до складних інформаційних та комунікаційних систем. У США 60% програмних продуктів написані на Visual Basic. Є звичайно дуже невеликі обмеження, наприклад, на Visual Basic не можна написати повноцінну DLL бібліотеку, драйвер пристрою VXD, також у Visual Basic не можна використовувати асемблер. Але коли вам знадобляться ці речі, ви зможете написати їх на іншій мові і, потім, використовувати їх у Visual Basic. Так що можливості Visual Basic дуже великі!

Тепер можна приступати до інсталяції та налаштування Visual Basic!

Урок № 3. Інсталяція та налаштування Visual Basic

Інсталяція Visual Basic не відрізняється особливою складністю, все відбувається стандартно. В процесі інсталяції вас попросять вказати компоненти, які будуть встановлені на ваш комп'ютер. Якщо на вінчестері є місце, то краще вибрати всі компоненти, щоб потім їх не довелося додавати. Варто відзначити той момент, що версія VB 5.0 поставляється з файлами допомоги (приблизно 15 Мб), а допомога для VB 6.0 йде в комплекті з MSDN, який поставляється на трьох дисках. Тому якщо у вас є шоста версія, записана на одному диску, то будьте впевнені, допомоги там не буде. Але цілком спокійно можна взяти допомогу від п'ятої версії і використовувати її для VB 6.0. Після інсталяції та налаштування Visual Basic обов'язково перезавантажите комп'ютер.

Отже, комп'ютер перезавантажений. Перед роботою з Visual Basic його необхідно налаштувати. Для цього запустіть Visual Basic (Пуск->Програми->Microsoft Visual Basic 6.0->Visual Basic 6). Зайдіть у меню Tools->Options. Поставте галочку "Require Variable Declaration". Це позбавить Вас від зайвих помилок при автоматичному визначенні змінних. Далі на вкладці Editor Format, у списку Font укажіть Courier New Cyr. Якщо цього не зробити, то VB не буде коректно відображати кирилицю. Також рекомендую установити колір зарезервованих слів у яскраво-синій. Для цього виберіть у списку Code Colors Keyword Text і в поле Foreground укажіть яскраво-синій колір (сьомий знизу). От і все! Visual Basic готовий до роботи!

Наступний урок призначений для тих, хто ніколи не програмував. Якщо ви знайомі з програмуванням, то можете сміливо пропустити цей урок.

Урок № 4. Для тих, хто ніколи не ...

Цей маленький урок для тих, хто ніколи не програмував. Хоча таких людей не існує! Адже ви програмуєте свою поведінку з огляду на різні варіанти подальшого ходу подій і змінюєте "програму" у залежності від обставин. Наприклад, якщо ви занедужали, то Ви напевно не підете на роботу (або в школу) і т.п. Вся відмінність життєвої програми від програмування на комп'ютері складається в рівні формалізації порядку дій, які необхідні для досягнення результату. Адже комп'ютер - це машина, вона не зрозуміє вас з півслова, більше того, він

узагалі вас не може розуміти! Він тільки виконує закладені в нього інструкції і команди для роботи з двійковими числами.

Для написання більш-менш складної програми необхідно спочатку скласти умовну послідовність дій на папері. Наприклад, “як зварити борщ?”:

Життєвий варіант програми:

Купити на ринку овочів (подешевше), сметани, все почистити, порізати, налити води, поставити на плиту і варити півгодини. Ще не забути посолити.

Формалізований варіант (Алгоритм):

Для всіх продавців овочів від першого до останнього

Якщо овочі в даного продавця дешевше, ніж у раніше перевірених, то записати його номер на папірець замість раніше на ньому записаного

Наступний продавець (див. п.2, якщо вони ще залишилися)

Купити овочі в продавця з номером, записаним на папірці

Купити сметану

Виконувати очищення овочів, поки видно шкірку та пошкодження

Збільшуємо шар, що зрізується, на міліметр

Повернення і перевірка умови в п.6

Для всіх куплених овочів від першого до останнього

Розділити плід на 40 частин

Наступний (див. п.10, якщо вони ще залишилися)

Помістити в каструлю. Налити води, включити газ

Таймер включити

Якщо таймер показує, що пройшло півгодини, то газ виключити

Якщо сіль відсутня в готовому продукті, то перейти до обробки помилки по п.16, інакше перейти до п.17

Ввести сіль

Кінець роботи програми

Простота написання програм на бейсік полягає в тому, що як команди він використовує англійські слова, які еквівалентні українським із формалізованого варіанта, тобто якщо перекладач, не знайомий з інформатикою, переведе його на англійський, то вийде програма на бейсік. Працювати вона не буде. Закон програмування каже: “Жодна, навіть найпростіша програма, не працює відразу після написання”. Будь-яку програму необхідно налагоджувати (debug (bug-жук) - обезжучувати). *На цей рахунок є ще один закон:* легше написати свою власну програму, чим розбирати та виправляти чужу. Налагодження програми по відчуттях нагадує прощтовхування важкої вантажівки по розмитій дорозі: у хід йдуть одні дошки, камені і лопата, і так доти, поки програма не почне правильно працювати. Якщо говорити мовою алгоритму, то вийде наступне:

Запустити програму

У випадку збою або неправильної роботи знайти причину помилки

Усунути помилку, намагаючись не внести нових

Продовжувати по п.1 доти, поки не будуть усунуті всі помилки

Якщо Ви засвоїли вищевикладений матеріал, то можете переходити безпосередньо до вивчення мови Visual Basic. А для цього переходимо до п'ятого уроку.

Урок № 5. Етапи розробки додатка

У Visual Basic, як і в багатьох інших мовах, призначених для написання додатків під Windows, використовується подійно-керована модель програмування. У “старих” версіях бейсика, таких як QBasic, використовувалася так звана “плоска” структура написання програми. Кожна програма починалася і закінчувалася у певних місцях. Вся програма виконувалася послідовно, і іноді, можливо, викликалися користувальницькі процедури і функції. Тобто, якщо програму “запускали”, то вона відразу починала виконуватися, і, дійшовши до кінця, завершувалася. У Visual Basic це відбувається зовсім інакше. Справа в тому, що ОС Windows має GUI (Graphical User Interface), тобто графічний інтерфейс користувача, у якому використовуються стандартні елементи керування, такі, як вікна (вони ж форми), кнопки, списки, поля для введення тексту і т.п. У будь-якій мові високого рівня програма будується на основі цих елементів. Отже, *розробка додатка на VB складається з наступних етапів:*

- продумати програму (продумати, що саме програма повинна робити, які саме задачі повинна вирішити, реалізувати їх подумки, продумати структуру даних, і т.д.).
- проектування інтерфейсу, тобто розміщення на формі потрібних керуючих елементів, кнопок, списків і т.п. Цей етап називається складанням кістяка програми.
- написання програмного коду, який поєднує розміщені на формі керуючі елементи, тобто “*нарощування плоті на кістяк*”.
- наладка програми. Цей етап часто займає більше часу, чим попередні.
- остаточна компіляція і, якщо це необхідно, створення дистрибутива (тобто інсталяційного файлу setup.exe).

Ці етапи завжди необхідно тримати в пам’яті, оскільки вони використовуються при написанні будь-якої програми. Порядок етапів теж важливий. Наприклад, не можна спочатку написати код програми, а тільки потім розробляти інтерфейс користувача (є звичайно рідкісні винятки, наприклад, коли програма взагалі не має інтерфейсу, або коли ви пишете процедуру, незалежну від інтерфейсу програми). Описувати перший етап не має смислу, тому що він цілком зрозумілий. Зупинимося докладніше на другому етапі.

Урок № 6. Структура проекту Visual Basic

Перед розглядом питання проектування інтерфейсу додатка на Visual Basic, необхідно уявляти, з чого взагалі складається цей проект?

У Visual Basic будь-який проект складається з наступних файлів:

- файл кожної форми (розширення frm). Це звичайний ASCII текстовий файл, у якому записаний весь код, розташований на формі, а також властивості всіх розміщених на формі елементів керування і самої форми теж.
- файл кожної форми, який містить бінарну інформацію (наприклад картинку в PictureBox) (розширення frx)
- файл проекту, який містить інформацію про проект (розширення vbp)
- інформація про робочу область проекту (workspace) (розширення vbw)

Це необхідний мінімум. (Хоча, бувають і винятки, наприклад, коли в проекті не використовуються форми. Тоді замість frm файлу, буде bas файл.)

Далі перелічимо додаткові файли, що можуть бути підключені до проекту:

- файл кожного модуля (розширення bas). Це текстовий файл.

- файл кожного модуля класів (розширення cls). Це текстовий файл.
- файл кожного додаткового елемента керування (розширення ctl). Це також текстовий файл.
- файл ресурсів (розширення res).
- інші файли (ocx, tlb, і т.д...).

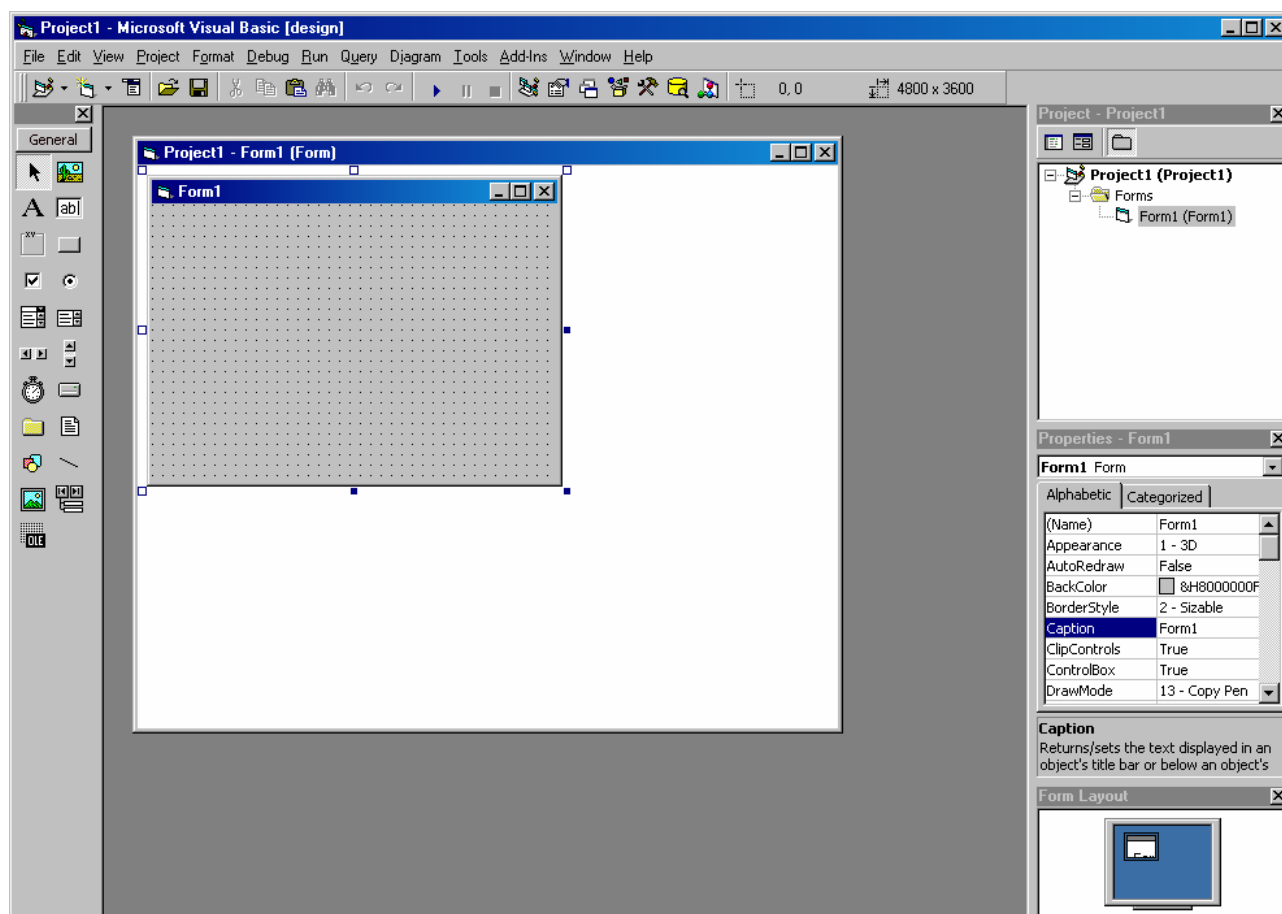
Запам'ятовувати призначення всіх цих файлів не обов'язково, *досить запам'ятати 2 файли:*

- frm-файл, у якому зберігаються код форми і властивості всіх розміщених на даній формі елементів керування.
- bas-файл - модуль. У ньому можуть бути оголошені глобальні змінні, константи, функції і т.д. Коротше, тільки код. Без елементів керування.

Тепер, приблизно уявляючи структуру проекту на Visual Basic, можна переходити до розгляду питання проектування інтерфейсу.

Урок № 7. Середовище розробки Visual Basic

Для того, щоб зрозуміти як проектувати інтерфейс, розглянемо спершу середовище розробки Visual Basic. Запустіть на виконання Visual Basic. Перед вами з'явиться віконце, у якому Вас попросять указати тип проекту. Укажіть тип "Standart EXE" і натисніть ОК. На екрані Ви побачите наступне вікно:



У лівій частині розташована панель з доступними елементами керування, з якої Ви можете перетаскувати потрібні елементи на форму. У центрі знаходиться форма (вікно) Вашого додатка. Ім'я нової форми - Form1. Угорі розташована панель інструментів середовища

розробки. Праворуч розташовані вікна проекту (Project) і властивостей поточного об'єкта (Properties). Тут необхідно відзначити, що всі об'єкти в Visual Basic (втім як і в інших мовах високого рівня) мають властивості і методи. *Властивості* — значення, що встановлюються для визначення виду і поведінки об'єкта. *Методи* — програмні процедури, що забезпечують виконання об'єктом деяких визначених дій. Наприклад, форма забезпечує метод Show, що обумовлює виведення форми на екран. Головна перевага роботи з об'єктами в тому, що об'єкти забезпечують програмний код, який уже не потрібно писати розробнику. Йому просто потрібно установити властивості об'єкта і викликати методи об'єкта, щоб примусити об'єкт виконати необхідні функції. Деякі властивості можна відредагувати тільки в період виконання програми (RunTime). Багато об'єктів мають однакові властивості. Наприклад, властивість Caption. У формі (Form) Caption - це заголовок вікна, а в елемента мітки (Label) - це текст усередині мітки. Поступово Ви звикнете до таких позначень і згодом зможете розібратися з кожним, незнайомим вам елементом керування.

Отже, для того щоб помістити на форму потрібний вам елемент, необхідно проробити наступне: натиснути мишкою на потрібний Вам елемент на панелі ліворуч, наприклад, на кнопку (Command Button). Після натискання кнопка виявиться втисненою. Тепер помістіть курсор миші на форму і розтягніть мишкою прямокутник. У результаті цих маніпуляцій у Вас на формі з'явиться кнопка, яка має розміри прямокутника. Якщо Вам знадобиться змінити розміри кнопки, то необхідно виділити кнопку натисканням лівої кнопки і розтягнути кнопку за маркери, розташовані на вершинах кнопки. Перемістити кнопку в інше місце можна звичайним перетаскуванням (Drag&Drop). Необхідно відзначити, що не всі елементи керування мають розміри. Наприклад, Timer. Такі елементи не видно в процесі роботи додатка, але вони виконують певні функції.

Щоб змінити властивості елемента керування необхідно виділити його та змінити потрібну властивість у вікні Properties. Давайте, приміром, поміняємо заголовок форми. Клацніть лівою кнопкою миші в будь-яке місце форми та знайдіть у вікні Properties властивість Caption. Змініть її, наприклад, на “Це моя перша форма”. Заголовок буде мінятися в міру введення тексту.

Тепер давайте запусимо програму. Для цього натисніть на кнопку Start, розташованої на панелі інструментів і кнопки, що має іконку Play (як на магнітофоні). Після нетривалої компіляції, перед вами з'явиться вікно вашої програми! Причому ви можете його пересувати, змінювати розміри, мінімізувати! І все це без єдиного рядка коду! Фантастика! :)

Тепер закрийте програму. Це можна зробити двома способами - натиснути на хрестик у правому верхньому куті форми або натиснути на кнопку End, яка має іконку кнопки Stop.

Після закриття програми Ви повернетесь в середовище розробки VB. До речі, зверніть увагу на віконце Project. У ньому показаний тільки один файл - Form1. Давайте додамо ще одну форму до нашого проекту. Для цього виберіть у меню Project->Add Form. Перед Вами з'явиться вікно Add Form, у якому вам запропонують вибрати вид нової форми. Двічі клацніть на іконці з написом Form. Перед вами з'явиться нова форма. Її ім'я Form2. Але куди ж ділася стара, запитаєте ви? Для того, щоб побачити нашу стару форму, потрібно двічі клацнути по рядку Form1 у вікні Project. Клацнувши, Ви відразу побачите нашу першу форму. Зверніть увагу на 2 кнопочки у вікні Project. View Object і View Code. Ці кнопочки Вам будуть в майбутньому дуже потрібні. За допомогою них Ви зможете переключатися між двома режимами:

- переглядом форми, для проектування її інтерфейсу
- переглядом коду форми

Натисніть на кнопку View Code. Якщо Ви налаштували Visual Basic, згідно рекомендацій, поданих в третьому уроці, то в коді Ви побачите тільки одку рядок:

Option Explicit

Цей рядок виділений синім кольором, тому що це зарезервоване слово Visual Basic. Якщо такий рядок є на початку модуля, то змінні не можна буде використовувати доти, поки вони не оголошені оператором Dim (як у С).

Тепер давайте збережемо наш проект. Для цього натисніть на кнопку Save Project (на панелі п'ята). Вас попросять вказати каталог для збереження й ім'я файлу для форми №1 (Form1.frm). Потім файл Form2.frm. І нарешті файл Project1.vbp (vbp - Visual Basic Project). Усі, проект збережений. Тепер ви зможете відкрити його надалі кнопкою Open Project. Відкривати потрібно файл із розширенням vbp (Visual Basic Project). Тут можу дати одну дуже важливу пораду - частіше зберігайте результати Вашої роботи! Все може бути... То з Windows проблеми, то світло вимкнуть...

Як бачите, нічого складного. Просто потрібно звикнути до інтерфейсу середовища Visual Basic. На цьому ми закінчимо цей урок. Як домашнє завдання, додайте до проекту модуль (Project->Add Module). І збережіть проект.

У наступному уроці Ви дізнаєтеся, як приємно редагувати код у Visual Basic.

Урок № 8. Легкість роботи з кодом у Visual Basic

У цій маленькій главі я б хотів розповісти про легкість роботи з кодом у Visual Basic.

Intellisense

Технологія *Intellisense* сильно полегшить вам життя в процесі програмування на Visual Basic. Ця технологія Microsoft дозволить вам уникнути введення великої кількості коду і його коректування. *Intellisense* виводить невелике спливаюче вікно з корисною інформацією про поточний об'єкт. Напевно ви вже бачили таке вікно. *Такі вікна бувають 3-х видів:*

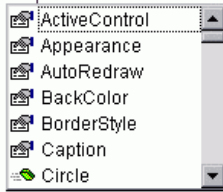
- *QuickInfo*. Видає інформацію про синтаксис поточного оператора Visual Basic. Де б ви не ввели ім'я оператора (функції) і поставили після імені пробіл або окриваюцю круглу дужку, то Visual Basic негайно покаже інформацію про синтаксис цього оператора. От приклад:

```
D = CalcDiscremenant (|
    CalcDiscremenant(a As Double, b As Double, c As Double) As Double
Tf D > 0 Then
```



- *List Properties/Methods*. Ця властивість полегшить вам роботу з об'єктами в Visual Basic. Після того, як ви поставите крапку після імені якого або об'єкта, Visual Basic відразу ж покаже вам список усіх доступних властивостей і методів цього об'єкта:

2. List Properties/Methods. Это свойство облегчит вам работу с объектами в Visual Basic. После того, как вы поставите точку после имени какого либо объекта, VB сразу же покажет вам список всех доступных свойств и методов этого объекта:

```
Private Sub Form_Load()
    Form1.
End Sub
```



Свойства имеют иконку: , а методы - .

Властивості мають іконку: , а методи - .

- *Available Constants*. Ця функція виводить вікно доступних констант. Наприклад, якщо Ви поставите знак рівності після Boolean змінної, то Visual Basic видасть вам вікно, де ви зможете вибрати з двох значень (True/False) потрібне. Вам навіть не прийдеться нічого набирати на клавіатурі!

Також, Якщо натиснути Ctrl+J, те Visual Basic видасть список усіх визначених у програмі властивостей, методів, констант, типів і т.д, включаючи убудовані в сам Visual Basic.

Також для новачків може бути корисна функція Auto Syntax Check, яку можна включити в Tools->Options. Якщо галочка коштує, то Visual Basic буде стежити за правильністю набраного коду в Visual Basic. Якщо ми наберете рядок невірно, то Visual Basic попередить вас про цьому, видавши вікно з повідомленням.

Ще Visual Basic постійно стежить за красою коду :). Оскільки Visual Basic не розрізняє великі і маленькі букви, він буде постійно коректувати імена **змінних** і функцій у кодї програми, щоб вони виглядали саме так, як зазначено в їхньому визначенні. Наприклад, якщо ви оголосите змінну

```
Dim myVar As String
```

А потім у кодї програми введете:

```
MYVAR = "VB"
```

То після того, як курсор редагування перейде на наступний (або просто інший) рядок, Visual Basic змінить код у такий спосіб:

```
myVar = "VB"
```

Тобто ім'я змінної завжди буде написано так, як визначено в операторі Dim. Це дуже хороша функція Visual Basic.

Також Visual Basic підсвічує кольорами певні ділянки коду. Синім кольором виділяються зарезервовані слова Visual Basic. Синьо-зеленим кольором (якщо його можна так назвати) коментарі, інше чорним. Коментарі - це усе, що знаходиться після символу ' (апостроф). При компіляції коментарі ігноруються, але при перегляді коду вони дуже корисні. Раджу коментувати код, особливо в тих місцях, де сам чорт зломить ногу! ;). Приклади підсвітки:

Dim prgVariable As Long ' це коментар

Відступи

Про відступи я б хотів би поговорити окремо. Відступи дуже важливі при програмуванні. Будь ласка, ніколи не забувайте про них! Вони допоможуть при перегляді Вашого коду. Особливо вони корисні в складних розгалуженнях і циклах. *Давайте розглянемо приклад:*

```
If Form1.Visible = False Then
  If a = b Then
    For c = 1 To 5
      If b > c Then Exit Sub
    Next c
  End If
End If
```

Погодьтеся, зрозуміти в такому коді що виконується після чого - досить складно. Але якщо ми поставимо відступи, то все відразу стане зрозумілим!:

```
If Form1.Visible = False Then
  &nbsp;&nbsp;&nbsp;If a = b Then
    &nbsp;  &nbsp;&nbsp;For c = 1 To 5
      &nbsp;  &nbsp;&nbsp;&nbsp;If b > c Then Exit Sub
    &nbsp;  &nbsp;&nbsp;&nbsp;Next c
  End If
End If
```

Як бачите, тепер відразу видно, що другий If виконається тільки при виконанні першої умови, що цикл знаходиться усередині розгалуження, і т.д.

Visual Basic надає можливість зробити відступ відразу для ділянки коду. Для цього необхідно виділити цю ділянку (кілька рядків) і натиснути Tab. Усі виділені рядки змістяться вправо. Якщо Вам необхідно змістити код уліво, натискайте Shift+Tab. Завжди пам'ятайте про відступи!

Далі ще цікавіше!

Урок № 9. З чого складається код?

Отже, ми навчилися проектувати інтерфейс програми. Але для повноцінної програми цього не досить. Потрібно написати код програми, що буде маніпулювати елементами керування і робити певні обчислення. Це самий складний етап.

В усіх мовах високого рівня програмний код складається з:

- Змінних
- Виразів
- Операторів
- Керуючих структур
- Функцій
- Класів і об'єктів

Опишемо кожен тип докладніше:

Змінні

У Visual Basic змінні зберігають інформацію (значення). При їхньому використанні Visual Basic резервує область у пам'яті комп'ютера для збереження даної інформації. Кожна змінна має своє ім'я. Воно може досягати 255 символів у довжину, починається завжди з букви латинського алфавіту, після якої можуть бути інші букви, цифри і знак підкреслення. Регістр символів значення не має. Приведемо кілька прикладів імен **змінних**:

numOfLetters - підходить

2Bottle - невірно, тому що починається не з букви

ThisIsVeryLongName - підходить, довжина 18 символів

sng.Cos - не підходить, тому що використовується крапка

Іменування змінних і функцій надзвичайно важлива справа. Я рекомендую дотримуватись угорської угоди:

Тип	Схема іменування	Приклад
Константа	Ім'я константи повинно складатись лише із заглавних букв	HWND_BROADCAST
Змінна	Ім'я змінної повинно починатись лише з маленької букви, а далі наступні слова – з вкликої	numOfFonts
Функція	Ім'я функції повинно починатись із заглавної букви, далі наступні слова також із заглавної	SetForegroundWindow

Постарайтеся щоб іменування змінних відповідно до такої схеми у Вас увійшло в звичку. Я раджу, але не наполягаю. Не подобається - будь ласка, називайте як Вам завгодно, от тільки так Ви ніколи не станете професійним програмістом.

Кожна змінна має визначений тип. *Всього Visual Basic 14 типів змінних.* Крім того, програміст може визначити і свій тип. *Перелічимо основні типи змінних VB:*

Byte - призначений для збереження цілих чисел від 0 до 255. Якщо змінній такого типу привласнити значення, що виходить за ці межі, то Visual Basic згенерує помилку.

Integer - призначений для збереження цілих чисел у діапазоні

-32768 до +32767, тобто розмір пам'яті, який виділяється під таку змінну складає 2 байти. (256*256=65536). Символ для позначення - "%". Навіщо він потрібний, ми розглянемо далі.

Long - призначений для збереження цілих чисел у діапазоні

-2147483648 до +2147483647, тобто розмір пам'яті, який виділяється під таку змінну складає 4 байти. (65536*65536=4294967296). Символ для позначення - "&".

String - призначений для збереження строкової (символьної) інформації, простіше кажучи - тексту. Може зберігати до 2 Гб тексти. Символ для позначення - "\$".

Single - призначений для збереження дробових чисел, з точністю до 7 цифр. Діапазон негативних значень від -3.402823E38 до -1.401298E-45. Діапазон позитивних значень від 1.401298E-45 до 3.402823E38. Довжина числа може досягати 38 знаків. Займає 4 байти пам'яті. Обчислення з даними змінними будуть приблизними і менш швидкими, чим із змінними цілого типу. Символ для позначення - "!".

Double - призначений для збереження дробових чисел, з точністю до 16 цифр. Діапазон негативних значень від 1.79769313486232E308 до -4.94065645841247E-324. Діапазон позитивних значень від 4.94065645841247E-324 до 1.79769313486232E308. Довжина числа може досягати 300 знаків. Займає 8 байтів пам'яті. Обчислення з даними змінними будуть приблизними і менш швидкими, чим із змінними цілого типу. Використовується для наукових розрахунків. Символ для позначення - "#".

Currency - Даний тип створений для того, щоб уникнути помилок при перетворенні чисел з десяткової форми в двійкову і навпаки (Неможливо представити 1/10 як суму 1/2, 1/4 і т.д). Даний тип може мати до 4 цифр після коми, і до 14 перед нею. Усередині даного діапазону обчислення будуть точними. Обчислення виконуються так само повільно, як і у випадку змінних Single і Double. Даний тип дуже добре підходить для фінансових розрахунків. Символ для позначення - "@".

Date - Цей тип даних дозволяє зберігати значення часу і дати в проміжку від напівночі 1 січня 100 року до напівночі 31 грудня 9999 року. Якщо змінній присвоюється тільки значення дати, то час дорівнює 00:00.

Boolean - дуже важливий і розповсюджений тип даних. Дозволяє зберігати так звані булеві значення, тобто тільки два значення - True і False. (українською - Правда і Неправда). Використовується тоді, коли Вам потрібно зберігати тільки значення **Так** чи **Ні**.

Variant - Змінна типу Variant може містити дані будь-якого типу. Visual Basic автоматично робить необхідні перетворення даних, тому не варто турбуватися про це. Використання такого типу даних сповільнює роботу програми, тобто потрібен час і ресурси для перетворення типів. Тому рекомендую завжди утримуватися від застосування цього типу даних, за винятком специфічних випадків, наприклад, повернення функцією масиву можливо тільки при використанні типу Variant.

У Visual Basic змінні оголошуються за допомогою оператора Dim, після якого проставляється As та Ім'я_Типу змінної. Наприклад:

```
Dim a As Long
Dim b As Byte
Dim c As Long
Dim numOfLetters As Long
Dim myString As String
Dim isLoading As Boolean
```

Якщо не вказувати As Ім'я_Типу, то змінна буде оголошена як Variant.

Після оголошення змінної їй привласнюється значення за замовчуванням. Для рядка це - "" (порожній рядок). Для чисел - 0. Для Boolean - False. Змінна може бути використана відразу після оголошення.

Змінні можна повідомляти й в одному рядку, розділяючи оголошення комами:

```
Dim a As Long, Dim b As Integer
```

Тут дуже важливо помітити наступну особливість. Логічно було б оголосити 3 змінні типи Long у такий спосіб:

```
Dim a, b, c As Long
```

Така звичка могла перейти, наприклад, з мови C. Там усі змінні дійсно мали б той тип, що зазначений після останньої змінної. Але не в Visual Basic! У результаті такого оголошення Visual Basic оголосить 3 змінні, перша і друга будуть мати тип Variant, і тільки третя - Long! Запам'ятаєте це! А взагалі, краще повідомляти кожну змінну в окремому рядку. І, якщо необхідно, те логічно відокремити ці оголошення можна просто вставивши порожній рядок між тими оголошеннями, що ви хочете відокремити логічно. Наприклад, так:

```
Dim a As Long
```

```
Dim b As Long
```

```
Dim myString1 As String
```

```
Dim myString2 As String
```

Такі логічні пробіли між рядками необхідні для рятування коду від його монотонності.

Привласнити значення змінної можна за допомогою знака дорівнює “=”. Наприклад:

```
a = 1234567
```

```
b = 234
```

```
c = 133
```

```
myString = “Visual Basic рулить”
```

```
isLoading = True
```

У даному прикладі змінним привласнювалися константи. Але часто буває необхідно привласнити одній змінній значення іншої. Це робиться в такий спосіб:

```
a = b
```

```
' можна, тому що змінна b “уміщається”
```

```
' у тип Long (234 < 4294967296)
```

```
a = b + c
```

```
' тепер у a зберігається сума b + c.
```

```
b = c
```

```
' теж можливо (133 < 255)
```

```
b = a
```

```
' не можна, тому що змінна a не укладається
```

```
' у діапазон [0-255]. Відбудеться помилка.
```

```
myString = a
```

```
' Visual Basic сам перетворить число 1234567
```

' у рядок "1234567", а потім привласнить це значення змінної myString. Також можливо зворотна дія.

```
isLoaded = True
' всі ОК
```

```
myString = myString & "однозначно!"
' тут відбувається злиття двох рядків, у результаті чого
' змінна myString містить рядок:
' "Visual Basic рулить однозначно!"
```

```
isLoaded = a
' можливо, тому що Visual Basic сам перетворить тип Long
' у тип Boolean. isLoaded буде містити True.
' Чому не False?
' Тому, що в VB False - це нуль, а будь-яке,
' не нульове значення - True
```

Вище, при описі типів змінних я вказував символ для позначення. Цей символ можна (а іноді і потрібно) використовувати для оголошення змінної, але без використання зарезервованих слів As Тип. Т.е. до приклада:

```
Dim myLongParam&
Dim myString$
```

Тут оголошені 2 змінні, перша має тип Long, друга - String. Також можна використовувати ці спеціальні символи для явного зазначення типу константам, наприклад:

```
Call MyProc (myParam1, myParam2, 5&)
```

Тут, при виклику процедури MyProc, останній параметр має тип Long. Якби ми не вказали значок &, то він (параметр) мав би тип Byte. Це необхідно при використанні API функцій.

Вище я вже відзначав, що Visual Basic досить часто, непомітно для розробника, займається перетворенням типів змінних автоматично. Приміром, ми розглядали такий приклад:

```
myString = a
```

Змінна **a** має тип Long, а myString - String. Visual Basic сам перетворить змінну **a** у тип String, а потім привласнить значення змінної MyString. Це необхідно знати. Також, Visual Basic надає у Ваше розпорядження кілька функцій перетворення типів: CLng, CBool, CDate, CStr і т.д. Кожна функція перетворить вираз до відповідного типу. Розглянемо розповсюджену помилку програміста на Visual Basic. Нехай у нас є код типу:

```
Dim a As Byte
Dim b As Byte
Dim c As Long
```

```
a = 200
b = 200
```

```
c = a + b
```

Здавалося б, що якщо запустити такий код на виконання, то в змінній *c* буде знаходитися значення 400 (200 + 200). Але ні! Visual Basic на рядку *c = a + b* згенерує помилку Overflow (Переповнення). Справа в тому, що у виразі праворуч від знака дорівнює складаються 2 змінні типи Byte, і Visual Basic вирішує, що після обчислення цього виразу має залишитися той же тип - Byte. Але якщо згадати те, що тип Byte може зберігати значення в діапазоні 0-255, можна зрозуміти чому Visual Basic генерує Overflow (Переповнення). 400 далеко виходить за діапазон Byte... Тут може виникнути питання: “А як же бути? Повідомляти змінні *a* і *b* типом Long?”. Можна зробити і так. А можна скористатися функцією перетворення типу CLng. Тоді працездатний код буде виглядати в такий спосіб:

```
Dim a As Byte
Dim b As Byte
Dim c As Long
```

```
a = 200
b = 200
```

```
c = CLng(a) + CLng(b) 'Усі в порядку, 400 - входить у тип Long
```

Трохи про константи

У Visual Basic можна оголошувати не тільки змінні, але і константи. Константа як і змінна, також зберігає певне значення, але на відміну від змінної збережене значення не може змінюватися. Щоб оголосити константу необхідно використовувати зарезервоване слово Const, за яким впливає ім'я і значення (і можливо тип) константи:

```
Const PI = 3.1415
```

В даному прикладі оголошена константа з ім'ям **pi** і значенням 3.1415. Після оголошення вона може бути використана по призначенню.

Можна відзначити наступну особливість Visual Basic: для констант із плаваючою крапкою тип за замовчуванням - Double, для цілих чисел - Integer. (Це легко можна перевірити вбудованою функцією Visual Basic - VarType). Для того, щоб явно задати тип константи, необхідно після імені задати тип, наприклад:

```
Const PI As Long = 3 ' PI = 3, PI має тип Long
```

У Visual Basic існує дуже багато вбудованих констант, які з легкістю можуть бути використані у Ваших програмах. Наприклад, константа vbNewLine - містить 2 символи, з ASCII кодами 13 і 10, тобто перехід на новий рядок. Список констант можна подивитися в Object Browser'е. Для його виклику необхідно натиснути F2, знаходячись у середовищі Visual Basic.

“Видимість” змінних:

Оголошувати змінні можна в самих різних місцях:

- *Всередині процедури (або функції).* У цьому випадку змінну буде “видно” тільки в кодї цієї процедури (або функції). Якщо Ви спробуєте звернутися до такої змінної всередині коду іншої процедури, то Visual Basic згенерує помилку.

- В самому верху коду форми, тобто відразу після оператора *Option Explicit*. Це місце називається розділом *General Declarations* (розділ Глобальних Оголошень). Такі змінні буде “видно” у будь-якому місці коду форми. Тобто у будь-якій процедурі (або функції) форми. Змінні в даному місці можуть бути оголошені за допомогою зарезервованих слів *Private* і *Public*. Розглянемо три визначення:

```
Dim myLocalVar1 As Byte
Private myLocalVar2 As Integer
Public myGlobalVar1 As Long
```

Перші два визначення абсолютно еквівалентні. Змінні оголошені в такий спосіб будуть видимі в будь-якому місці коду форми. Але тільки тієї форми, де вони оголошені. В інших формах звернутися до таких змінних не можна.

Третя змінна буде видима всьому додаткові в будь-якому місці. Правда, щоб добратися до такої змінної з коду іншої форми, необхідно перед ім'ям змінної вказати ще й ім'я форми, де ця змінна оголошена, наприклад:

```
Form1.myGlobalVar1 = 234
```

- У розділі *General Declarations* модуля. Тут діють ті ж правила, що й у розділі *General Declarations* форми. *Private* (або *Dim*) будуть “видимі” тільки в кодї модуля. А *Public* - скрізь. Відмінність спостерігається тільки в способі доступу до змінної. Тут не обов'язково вказувати ім'я модуля перед такою змінною. Можна просто вказати її ім'я – і цього буде достатньо. Хоча я не рекомендую цього робити, тому що втрачається наочність. І, до того ж, якщо у Вас є два модулі, у яких оголошені змінні з однаковими іменами, то добратися до них можна буде тільки вказавши ім'я відповідного модуля перед ім'ям змінної.

Період існування змінних

Період існування змінної означає, як довго змінна є доступною.

- Змінні, оголошені в процедурі (або функції) будуть “живі” рівно стільки часу, скільки виконується конкретна процедура (або функція). При виході з процедури - змінна видаляється. При черговому виклику цієї процедури - змінна заново ініціалізується. До речі, до слів *Private*, *Public* і *Dim*, у процедурах і функціях можна використовувати зарезервоване слово *Static*. Така змінна при повторному виклику цієї процедури не буде заново ініціалізуватися. Вона буде зберігати те значення, що було в ній після попереднього виклику. (такі змінні дуже зручно використовувати в обробці події *Timer*'а). Наприклад:

```
Static myStat As String ' Private Static змінна
```

- Змінні рівня форми будуть “живі” тільки поки “жива” форма. Як тільки об'єктна змінна форми буде встановлена в *Nothing* (або після виконання оператора *Unload*), всі змінні рівня цієї форми видаляються.

- Змінні рівня модуля “живі”, поки “живе” Ваший додаток. Тобто “живі” завжди.

Певні зауваження:

- *Перед використанням змінної її обов'язково потрібно оголосити.* Це позбавить Вас від зайвих помилок.

- *Змінним потрібно давати осмислені імена.* Тобто намагайтеся не використовувати імена типу a1, a2, a3, ab, ccc і т.п. Повірте, рано чи пізно Ви заплутаєтеся у своїй же програмі! Найкраще змінним давати англійський еквівалент того, що зберігає дана змінна. Наприклад, для збереження певної суми грошей, змінна може бути названа як cashMoney. Якщо у Вас з англійським не дуже, то можете використовувати трансліт, тобто змінну назвати, наприклад як summaDeneg. Ну і т.п. Кирилицю в імені змінної використовувати не можна.

Переходимо до наступного уроку.

Урок № 10. Масиви, записи і перерахування

В попередньому уроці ми розглянули з вами таку важливу річ, як змінні. А змінні – це, образно кажучи, шматочки пам'яті, де зберігаються дані. Отже, якщо ми ефективно використовуємо змінні - ми ефективно використовуємо пам'ять. А якщо ми ефективно використовуємо пам'ять - то пам'яті для реалізації додатка потрібно менше і додаток працює швидше. Отже для того, щоб ці дані використовувати з максимальною ефективністю, і в той же час з легкістю, були придумані “масиви” (Arrays), “записи” (Types) і “перерахування” (Enums).

Масиви

Їх ще називають списками. Отже що ж таке масиви? Масив (вектор) - це набір однотипним змінним, об'єднаним одним ім'ям і доступних через це ім'я і порядковий номер змінній в наборі. Кількість елементів масиву теоретично може бути нескінченною, обмеження накладаються конкретною мовою програмування та операційною системою. Елементи масиву мають безперервну нумерацію певного діапазону.

У програмуванні масиви використовуються досить часто. Наприклад, вам потрібно завантажити в програму вміст якого-небудь файлу. Якщо цей файл строковий, то можна використовувати рядок і обійтися без масиву. А от якщо файл бінарний (двійковий), то без масиву просто не обійтися!

У Visual Basic масиви визначаються в такий спосіб:

```
Dim myArray (10) As Long
```

Як Ви вже могли помітити, визначення масиву відрізняється від визначення звичайної змінної тільки індексом, зазначеним у дужках. Цей індекс вказує розмірність масиву. У даному випадку масив myArray буде містити 11 елементів. Чому 11? Тому що нижня границя масиву починається з нуля. [0,1,2.....9,10]. Щоб задати певну розмірність можна використовувати зарезервоване слово **To**:

```
Dim myArray (5 To 10) As Long
```

Тут визначається масив, розмірність якого 6 елементів (5,6,7,8,9,10).

Загальний синтаксис визначення масиву наступний:

```
Dim ім'я масива{Номперв1 To Номпосл1, Номперв2
  Те Номпосл2, ...} [As [New] ім'я типа]
```

Багатомірні масиви

Масиви можна робити багатомірними. Наприклад, оголосимо масив - таблицю поля шахівниці:

```
Dim chessTable (1 To 8, 1 To 8) As String
```

Цей масив являє собою таблицю з восьми осередками по вертикалі і горизонталі.

Отже, масив визначений. Тепер необхідно довідатися - як же можна добратися до елементів цього масиву. Дуже просто! До елементів масиву потрібно звертатися по індексу, приміром, щоб змінити нульовий елемент масиву `myArray` потрібно написати:

```
myArray(0) = 1234
```

Або, наприклад:

```
chessTable (2,3) = "Пішак"
```

Масиви змінної розмірності (динамічні)

Динамічні масиви - це такі масиви, розмірність яких може мінятися в ході роботи програми. Мабуть динамічні масиви використовуються навіть частіше статичних. Розглянемо характерний приклад використання такого масиву. Нехай у нас є процедура, яка завантажує вміст двійкового файлу в масив. *Масив ми можемо визначити так:*

```
Dim fileContent (119) As Byte
```

Але це якщо файл має довжину 120 байт. А що робити, якщо ми не знаємо довжину файлу, який завантажується? Визначити дуже великий масив, щоб уже напевно туди міг поміститися великий файл? Немає. Так робити не можна. Саме в такій ситуації і потрібно використовувати динамічний масив.

Visual Basic надає досить могутні засоби для роботи з такими масивами. Визначається такий масив в наступний спосіб:

```
Dim myArray () As Byte
```

На відміну від масивів статичних розмірів, коли звертатися до елементів можна відразу після його оголошення, до елементів динамічного масиву відразу звертатися не можна, тому що вони ще не ініціалізовані. Для початку потрібно вказати його нову розмірність. Для цього Visual Basic має оператор `ReDim`. Працює він у такий спосіб:

```
ReDim myArray (4)
```

Такий масив `myArray` має одну розмірність з індексами від 0 до 4 (тобто всего 5 елементів). Тепер до такого масиву можна звертатися точно так само, як і до статичного. Якщо надалі виникне необхідність знову змінити розмірність масиву, можна ще раз використати `ReDim`.

Але, - тут є підводний камінь! Давайте розглянемо маленький приклад:

```
Dim myLong As Long
Dim myArray() As Long ' оголошуємо масив
```

```
ReDim myArray (2) ' одна розмірність [0,1,2]
myArray (1) = 234 ' привласнюємо другому елементу чило 234
myLong = myArray (1) ' зберігаємо його в змінній myLong
```

```
ReDim myArray (3) ' знову змінюємо тепер [0,1,2,3]
myLong = myArray (1) ' знову намагаємося зберегти другий елемент
```

На останньому рядку, змінній myLong буде привласнено значення 0 замість 234! Це відбувається тому, що оператор ReDim заново ініціалізує (скидає) всі елементи масиву до значення за замовчуванням (як пам'ятаєте, для чисел - це 0, для рядків ""). Але як же бути, якщо ми хочемо змінити розміри масиву, зберігши всі старі елементи? Для цього потрібно після оператора ReDim поставити слово Preserve. Приблизно так:

```
ReDim Preserve myArray (3) ' зберігаємо старі елементи
myLong = myArray (1) ' усе в порядку
```

Тепер усе в порядку.

Корисні ради по роботі з масивами в Visual Basic

Масиви можуть зберігатися в змінних типу Variant. Іноді це буває зручним. У деяких випадках без цього просто не обійтися! (Наприклад, коли Ви хочете, щоб ваша функція повертала масив). Щоб зберегти який-небудь масив у змінній типу Variant необхідно просто привласнити цій змінній потрібний масив:

```
Dim myVariantArray ' змінна Variant за замовчуванням
myVariantArray = chessTable
```

Зверніть увагу, ніякі індекси вказувати не потрібно!

Тепер можна використовувати копію як звичайний масив:

```
myVariantArray (0) = "Це копія"
```

Якщо Вам буде потрібно в коді програми довідатися поточні розміри масиву, то можна використовувати вбудовані функції Visual Basic - LBound і UBound. Перша функція повертає нижню границю масиву, друга верхню. Докладніше про ці функції читайте в довіднику (vbhelprus).

Записи

Ті, хто програмував на інших мовах програмування (таких, як C і Pascal), напевно зіштовхувалися з поняттям структура (C), і записом (Record у паскалі). У Visual Basic аналогом структури є запис. **Запис** - це новий тип даних, який визначається і складається з однієї чи більшою кількістю змінних усередині. Давайте розглянемо це на прикладі. Наприклад, необхідно в програмі зберігати масив студентів. Причому кожен студент має свої

характеристики: ПІБ, Вік, Наявність Грамот. Звичайно, для збереження таких даних можна використовувати, наприклад, масив, який має дві розмірності. Але це не кращий варіант. Найкраще тут підходять Записи! Потім із запису можна буде зробити масив! Щоб визначити запис у програмі потрібно використовувати зарезервоване слово Type. Закінчується запис словами End Type:

```
Private Type Student ' замість Private могло бути і Public
    &nbsp;FIO As String
    &nbsp;Age As Byte
    &nbsp;HasGramot As Boolean
End Type
```

Зауважте, що Dim перед ім'ям змінної вказувати не потрібно. Отже, ми визначили запис у програмі. Тепер можна оголошувати змінні, які мають тип - Student (тобто наш новий запис). Наприклад:

```
Dim newStud As Student
```

Слово Student синім виділятися не буде, тому що синім підсвічуються мають тільки зарезервовані слова, вбудовані в Visual Basic

Тепер, до полів запису можна звертатися за допомогою крапки:

```
newStud.FIO = "Віталій Євгенович Голиш"
newStud.Age = 19
newStud.HasGramot = False
```

Всі як у Паскалі. (ну, і майже як у C).

Visual Basic надає можливість не вказувати щораз ім'я змінної типу запис, при звертанні до її елементів. Це особливо корисно, коли запис має багато внутрішніх членів. Для цього є слово With:

```
With newStud
    .FIO = "Олександр Павлович Швардак"
    .Age = 20
    .HasGramot = True
End With
```

Прийшов час оголосити масив елементів типу запис (точніше типу Student):

```
Dim myStudArray (20) As Student
```

Тут ми оголосили масив з 21 студента. Тепер можна звертатися до елементів масиву точно так само, як ми це робили раніш:

```
myStudArray(0).FIO = "Оксана Іванівна Зеленьак"
```

Як бачите все геніальне просто! Особливо в Visual Basic! :)

Перерахування

Перерахування теж досить важлива і потрібна справа. В принципі, Ви з ними вже зустрічалися. Де? А згадайте, що відбувалося, коли Ви навпроти Boolean змінної ставили знак рівності? Правильно, Visual Basic видавав вам список із двох значень на вибір - True і False. Це і є перерахування. Іншими словами *перерахування* - це список констант. Перед використанням такого списку його необхідно визначити в програмі. Наприклад, розглянемо перерахування оцінок, які отримують студенти:

```
Enum Ocenka
    &nbsp;&nbsp;&nbsp;Neud = 3
    &nbsp;&nbsp;&nbsp;Horosho = 4
    &nbsp;&nbsp;&nbsp;Otlichno = 5
End Enum
```

Присвоювати значення константам всередині Enum не обов'язково. Якщо цього не зробити, то константи будуть приймати значення 0,1,2... і т.д.

Тепер можна оголосити змінні типу Ocenka:

```
Dim oc1 As Ocenka
```

І, якщо Ви тепер спробуєте присвоїти таке змінне значення - Visual Basic видасть список (Neud, Horosho і Otlichno) з якого ви зможете вибрати потрібне значення. Також ці константи можна використовувати, наприклад, при перевірці умов, тобто If oc1 = Horosho Then ... Але про умови пізніше.

Щоб закріпити отримані знання, давайте перевизначимо наш запис Student на наступну:

```
Private Type Student ' замість Private могло бути і Public
    &nbsp;&nbsp;&nbsp;FIO As String
    &nbsp;&nbsp;&nbsp;Age As Byte
    &nbsp;&nbsp;&nbsp;HasGramot As Boolean
    &nbsp;&nbsp;&nbsp;ikzamenFizika As Ocenka
    &nbsp;&nbsp;&nbsp;ikzamenMatan As Ocenka
End Type
```

Тепер щоб присвоїти оцінку 5 по матаналізу студенту під номером 3, необхідно написати:

```
myStudArray(2).ikzamenMatan = Horosho
```

Тепер, як бачите, код дуже прозорий. Такий код набагато легший для розуміння, тому перш ніж визначати які-небудь дані у своїй програмі, спочатку подумайте, як це краще реалізувати? Може краще щось зробити за допомогою записів? Або перерахувань? І т.д.

Примітка: Ті, хто програмував на паскалі пам'ятають про множини. Отож, може Вас засмутити, - множин у Visual Basic немає. Але в принципі, ніхто не заважає Вам реалізувати їх самостійно, написавши відповідні функції.

Ну от і все. На цьому закінчимо з масивами, записами і перерахуваннями. Настав час переходити до більш практичного – до виразів!

Урок № 11. Вирази

Отже, зі змінними ми розібралися. Тепер займемося виразами.

Вирази

Для початку розберемося з поняттям “*вирази*”. У будь-якій мові програмування вирази є основними цеглинками, з яких будується програма. Відповідно до самого точного визначення, що мені приходилося бачити, “*вирази*” - це “щось, що містить певне значення”. За прикладом далеко ходити не потрібно, візьмемо приклад з попереднього уроку:

```
b = 234
```

Тут ми привласнюємо змінній *b* значення 234. Іншими словами “234” - це вираз зі значенням 234. А тепер, для прикладу розглянемо рядок:

```
c = b
```

Тут змінній *c* привласнюється вираження *b*. Значення цього виразу - *b* = 234. Тобто іншими словами *b* - вираз, зі значенням 234. Розглянемо більш складний приклад вираз-функцію. Оголосимо функцію *MyFunc*, яка повертає байт 234:

```
Public Function MyFunc() As Byte
    &nbsp;MyFunc = 234
End Function
```

А тепер запишемо рядок:

```
c = MyFunc()
```

Як Ви вже напевно здогадалися, вираз тут - *MyFunc()*, зі значенням 234. Тобто після присвоєння змінній *c* виразом *MyFunc()*, вона буде містити значення 234. А от ще приклад:

```
c = 5 + 5 * 2
```

Тут вираз це $5 + 5 * 2$. Значення цього виразу не важко порахувати, воно дорівнює 15 (не 20!). Можна було б написати і так:

```
c = MyFunc() - 219
```

Тут значення виразу таке ж, як і в попередньому випадку, але от самі вирази різні. Це важливо розуміти.

Також необхідно відзначити значення дужок у виразах. Пам’ятаєте дужки в школі? Отож у Visual Basic дужки виконують ту ж функцію, що й у школі, а саме - задають пріоритет операції. Приміром, модифікуємо вираження $5 + 5 * 2$ на:

```
c = (5 + 5) * 2
```

Тепер значення цього виразу не 15, а 20! Запам’ятаєте це.

Йдемо далі. Розглянемо інші, дуже розповсюджені вирази - вирази порівняння. Такі вирази повертають True або False, у залежності від значення виразу, яке отримали. До таких виразів відносяться:

```
Dim bRes As Boolean ' змінна для збереження результату
```

```
Dim a As Long
```

```
Dim b As Byte
```

```
Dim c As Long
```

```
a = 234 ' змінні для тесту
```

```
b = 5
```

```
c = 1000
```

```
bRes = c > b ' 1
```

```
bRes = c < b ' 2
```

```
bRes = a >= c ' 3
```

```
bRes = b <= 4 ' 4
```

```
bRes = b <= 5 ' 5
```

```
bRes = a <> b ' 6
```

```
bRes = Not (a = b) ' 7
```

```
bRes = c = MyFunc() ' 8
```

```
' і їм подібні...
```

Отже, розглянемо кожен випадок докладно:

- Змінній типу Boolean - bRes привласнюється значення виразу (c > b). Тобто Visual Basic порівнює ці змінні, і дивиться, c > b? Якщо так, то значення цього виразу порівняння - True. Якщо ж ні, то False.

- Те ж саме, тільки bRes буде містити значення False, тому що c більше b, а не менше, як зазначено у виразі.

- Відбувається порівняння значень змінних a і c. Тобто a більше або дорівнює c? У нашому випадку a = 234, c = 1000, значить не більше і не дорівнює (менше). bRes буде містити False. b порівнюється з числом 4. Згадайте, адже 4 - це також вираз, зі значенням 4! Тобто b > 4, то bRes = False.

- Також відбувається порівняння змінної b з числом, 5. Але цього разу одна з умов виконання, а саме b = 5! Значить bRes = True.

- Очевидно, що b <> a. Отже bRes повинне дорівнює True!? Так, так воно і є... ;)

Зверніть увагу на цей приклад. Після виконання цього рядка, bRes буде дорівнює True! Тут вираз Not (a = b) обчислюється в такий спосіб: спочатку Visual Basic порівнює значення a і b. Після того, як VB переконається в тому, що a <> b (тобто False), він обчислює вираз:

Not (False). Оператор Not - це булів оператор заперечення. Він інвертує значення. У даному випадку з False виходить True (на більш низькому рівні, можу додати, що оператор Not інвертує всі біти операнда, у даному випадку, тому що False у VB - це 0, а True - FFFF, то значення виразу - True).

- В даному випадку нічого особливо не відбувається. bRes = False. Чому? Нехай це буде вашим домашнім завданням :).

Ну й останній приклад, що ілюструє застосування виразів порівняння. Зробимо так, щоб змінній a присвласнилось значення 234, якщо

```
c = b, і 100, якщо c <> b:
```

```
If c = b Then a = 234
```

```
If c <> b Then a = 100
```


Пам'ятаєте на попередньому уроці ми використовували Not, для заперечення . Отож - Not - це теж оператор, тільки не арифметичний, а логічний.

Оператори бувають різні:

У Visual Basic оператори бувають наступних типів:

Арифметичні:

^ оператор піднесення в степінь.

* оператор множення.

/ оператор ділення

\ оператор цілочисленого ділення

Mod оператор обчислення залишку від ділення

+ оператор додавання

- оператор віднімання

Порівняння:

< менше

> більше

<= менше або дорівнює

>= більше або дорівнює

= дорівнює

<> не дорівнює

Конкатенації:

+ оператор конкатенації

& оператор конкатенації

Логічні:

And оператор логічного множення

Eqv оператор логічної еквівалентності

Imp оператор логічної імплікації

Not оператор логічного заперечення

Or оператор логічного додавання

Xor оператор логічного додавання, що виключає

Як я вже говорив, кожний оператор має свій пріоритет, і кожний вираз обчислюється з урахуванням цих пріоритетів. Для довідки я хочу привести дуже цікаву таблицю пріоритетів:

Арифметичні	Порівняння	Логічні
Піднесення в степінь (^)	Рівно (=)	Not
Заперечення (-)	Нерівно (<>)	And
Множення та ділення (*, /)	Менше чим (<)	Or
Цілочислене ділення (\)	Більше чим (>)	Xor
Залишок від ділення (Mod)	Менше або рівно (<=)	Eqv
Додавання та віднімання (+, -)	Більше або рівно (>=)	Imp
Конкатенація (&)	Відповідність масці (Like)	
	Приналежність до типу (Is)	

В цій таблиці зверху вниз показаний порядок проходження пріоритетів операторів Visual Basic. Зверху вниз пріоритет зменшується. Оператори (* і /) мають однаковий пріоритет, і у виразі обчислюються зліва на право. Це ж саме відноситься до операторів додавання та віднімання. Якщо ж у виразі зустрічаються оператори з різних категорій, то обчислюються вони в порядку стовпчиків зліва на право. Тобто спершу арифметичні, потім оператори порівняння і лише потім логічні оператори.

Отже, йдемо далі. Нас чекають керуючі структури мови Visual Basic.

Урок № 13. Керуючі структури

Цей урок присвячений операторам, які призначені для керування процесом виконання всіх інших виконуваних операторів Visual Basic. Такі класичні структури, як умовні оператори й оператори циклу, є у всіх процедурних мовах програмування, - є вони й у Visual Basic.

Примітка: Варто відзначити, що керуючої структури ми також будемо називати операторами. Оператори, розглянуті на попередньому уроці використовуються у виразах. А оператори розглянуті в цьому уроці призначаються для керування обчисленням цих виразів (у документаціях такі оператори називаються - Statement). Важливо розуміти розходження між цими операторами.

Умовний оператор If...End If

Цей оператор Ви вже зустрічали у восьмому уроці. Він необхідний для прийняття рішень, чи потрібно виконувати ту або іншу дію чи ні. Іншими словами якщо Логічний_вираз істинний, то Оператор виконається. Якщо Логічний_вираз неістинний, - то виконання не відбудеться.

If Логічний_вираз **Then** Оператор

або складніше

```
If Логічний_вираз Then
    Група_операторів
End If
```

У першому випадку оператор може бути тільки один. В другому - скільки завгодно (у тому числі і один).

Приклад:

```
If (a = b) And (c <> d) Then
    &nbsp;   b = d
    &nbsp;   a = 20
End If
```

Дужки тут не обов'язкові, але вони підвищують читабельність коду.

Умовний оператор If...Else...ElseIf...End If

Така конструкція використовується для більш складних розгалужень:

```
If Логічний_вираз 1 Then
    Група_операторів
ElseIf Логічний_вираз 2 Then
    Група_операторів
...
Else
    Група_операторів
End If
```

Ця схема може бути й в скороченому виді **If...Then...Else...End If**. При цьому оператори після Else виконуються тільки в тому випадку, якщо жодна з умов не виконано.

Приклад:

```
If (a = b) Or (c <> d) Then
    &nbsp;   b = d
    &nbsp;   a = 20
Else
    &nbsp;   c = d
End If
```

Вираз c=d буде виконано тільки в тому випадку, якщо a<>b або c=d.

Умовний оператор **Select Case...End Select**

Конструкція Select Case "приймає рішення" на основі аналізу значення одного виразу. При цьому цей вираз вказується в рядку Select Case:

Select Case Вираз_для_аналізу

```
Case Значение№ 1
    Група операторів
Case Значение№ 2
    Група операторів
...
Case Значение№ N
    Група операторів
Case Else
    Група операторів
```

End Select

Цілком зрозуміло, що вираз для аналізу мусить повертати значення типу, сумісного з типом значень у рядок Case.

Приклад:

У залежності від значення змінної **iTest**, строковій змінній **strResult** присвоюються різні значення

Select Case iTest

```
Case 1
    &nbsp;   strResult = "iTest = 1"
Case 2, 3, 4
    &nbsp;   strResult = "iTest = 2, 3 або 4"
Case 5 To 9
    &nbsp;   strResult = "iTest знаходиться в діапазоні від 5 до 9"
Case iTest < 0
    &nbsp;   strResult = "iTest менше 0"
Case Is > 9
```

```

    &nbsp; strResult = "iTest більше 9"
Case Else
    &nbsp; strResult = "iTest дорівнює 0"

End Select

```

Оператор циклу For...Next

Цей цикл використовують у тому випадку, коли заздалегідь відоме стартове та кінцеве значення лічильника. Синтаксис виглядає в такий спосіб:

```

For Лічильник_циклу = Старт To Старт Step Крок
    Група операторів
Next [Лічильник_циклу]

```

Роль лічильника циклу може відігравати тільки раніше оголошена змінна цілочисленного типу. Крок задає збільшення лічильника циклу при кожному проході. За замовчуванням значення кроку дорівнює 1. Після слова Next лічильник можна опустити.

Приклад:

У цьому прикладі всім елементам масиву iArray присвоюється значення 5.

```

Dim c As Integer
Dim iArray(10) As Integer
For c = 0 To 10
    &nbsp;  iArray(c) = 5
Next c

```

Оператор циклу For Each...Next

Ця специфічна форма циклу For призначена для виконання певної операції з кожним об'єктом, який входить до складу певної колекції об'єктів (такою операцією, наприклад, може бути виклик методу або присвоєння значення властивості). Синтаксис оператора:

```

For Each Ім'яОб'єкта In Ім'яКолекції
    Операції над об'єктами
Next Ім'яОб'єкта

```

Приклад:

У цьому прикладі показано, як змінити властивість BackColor у всіх етикеток (Label), що лежать на формі

```

Dim x As Object
For Each x In Me.Controls
    &nbsp;  If TypeName(x) = "Label" Then
        &nbsp;  x.BackColor = 0
    &nbsp;  End If
Next x

```

Me тут - поточна форма. Тобто, не обов'язково використовувати повне ім'я форми для доступу до її властивостей. Наприклад, для закриття поточної форми, можна написати Me.Hide. (або Unload Me).

Оператор циклу **Do While...Loop / Do...Loop While**

Ці два різновиди циклу тісно взаємозалежні, і їх часто розглядають як один з базових видів циклу. Як уже відзначалося, цикли **For** застосовують у тих випадках, коли кількість проходів і діапазон зміни лічильника циклу заздалегідь відомі. Цикли **While** призначені для ситуацій, коли кількість проходів циклу заздалегідь не відомо, але зате відома умова виходу з циклу.

Синтаксис циклу While:

Do While Умова_виходу

Група операторів

Loop

Do

Група операторів

Loop While Умова_виходу

Відмінність між ними полягає в тому, що умова виходу перевіряється в одному випадку перед черговим проходом, а в іншому випадку - після виходу. Якщо в циклі пропустити умову виходу або ця умова завжди виконується, то отримаємо нескінченний цикл.
Наприклад такий:

```
Do While 2 > 1
    &nbsp;  Debug.Print "Вічний цикл"
Loop
```

Якщо у Вас випадково вийшов такий цикл, то вийти з нього можна при натисканні Ctrl+Break. Але це працює тільки в середовищі розробки.

Приклад:

```
Dim n As Integer
n = 100
Do While n >= 0
    &nbsp;  n = n - 1
    &nbsp;  Debug.Print n
Loop
```

Оператор циклу **Do Until...Loop / Do...Loop Until**

По своїй логіці цикл Until подібний циклу While з тією лише різницею, що проходи циклу виконуються доти, поки умова виходу не виконується.

Приклад:

```
Dim n As Integer
n = 100
Do
```

```

    &nbsp; n = n - 1
    &nbsp; Debug.Print n
Loop Until n < 11

```

Вихід з циклу Exit For / Exit Do

За допомогою операторів Exit For/Exit Do можна здійснити достроковий вихід з циклу поза залежністю від значення, що має в даний момент умова виходу.

Приклад:

```

Dim n As Integer
n = 10
Do While n > 1
    &nbsp; n = n - 1
    &nbsp; Debug.Print n
    &nbsp; If n = 5 Then Exit Do ' Якщо лічильник = 5, те
                        ' виходимо з циклу
Loop

```

Отже, *керуючі структури* – це дуже важлива і далеко не сама слабка ланка в програмуванні на Visual Basic (та й не тільки на Visual Basic). Без використання таких структур Ви не зможете написати навіть саму маленьку програму. Навіть якщо і напишете, то програма не буде являти собою ніякого практичного інтересу.

Але є ще більш важлива ланка програми - *функція*. От саме функціями ми зараз і займемося! Розглянувши функції, ми вже зможемо написати корисну програму.

Урок № 14. Процедури і функції

Вирази та оператори - це сировина для блоків, з яких будуються програми, де в ролі блоків виступають процедури і функції.

Процедури і функції. У Visual Basic, як і в багатьох інших мовах програмування, більшість програм створюється з блоків - процедур і функцій. Весь програмний код знаходиться як би усередині цих процедур. Якщо виникає необхідність у рішенні певної задачі в будь-якому місці програми, то викликається процедура. У Visual Basic не можна ввести код між процедурами. Код завжди мусить знаходитись усередині процедури.

Давайте розберемося з поняттями, і визначимо, що буде називатись процедурою, а що функцією.

Процедури

Процедура - це певний блок коду, який буде виконуватися щоразу при виклику цієї процедури. Кожна процедура починається зарезервованим словом **Sub** і закінчується **End**. *От загальний синтаксис процедури:*

```

[Private | Public | Friend] [Static] Sub name [(arglist)]
    [тут якийсь код]
Exit Sub
    [тут теж може бути якийсь код]
End Sub

```

Все, що вкладено в квадратні дужки - є необов'язковим. Оператор **Exit Sub** дозволяє достроково вийти з процедури. Іноді це дуже зручно. Слова **Public**, **Private** мають те ж значення, що і при оголошенні змінних.

arglist має такий вигляд:

```
[Optional] [ByVal | ByVal] [ParamArray] varname[( )]
[As type] [= defaultvalue]
```

Розглянемо приклад процедури, яка буде виводити на екран повідомлення "Hello World!":

```
Private Sub ShowMessage()
    &nbsp;MsgBox "Hello World!"
End Sub
```

Що можна сказати про цю процедуру? Процедура має тип **Private**, тобто доступна буде тільки з коду саме цієї форми (модуля), де вона оголошена (згадайте типи оголошення змінних). Дана процедура не містить параметрів, про що нам говорять порожні круглі дужки. Призначення - вивести повідомлення **Hello World** на екран.

MsgBox - це вбудована функція **Visual Basic**, яка виводить на екран вікно з повідомленням, заданим як параметр. Інші параметри необов'язкові (їх всього 5).

Як викликати процедуру? Для цього досить написати ім'я процедури:

ShowMessage

А можна і так:

Call ShowMessage' більш наочний варіант

Обидва цих варіанта абсолютно еквівалентні. Але для виклику процедур усе-таки краще використовувати другий варіант.

Тепер давайте змінимо цю процедуру і додамо до неї параметр, значення якого буде виводитися функцією **MsgBox** (замість **Hello World**):

```
Private Sub ShowMessage(message As String)
    &nbsp;MsgBox message
End Sub
```

Тепер при виклику процедури необхідно вказати параметр:

Call ShowMessage ("Наша перша процедура")

Результатом виконання такої процедури буде виведення на екран повідомлення: "Наша перша процедура". Дужки, що оточують параметр обов'язкові, якщо перед ім'ям процедури стоїть оператор **Call**. Якщо **Call** відсутній, то дужки ставити не потрібно.

Давайте розглянемо докладніше що ж відбувається при виклику нашої процедури. Зустрівши рядок з викликом нашої процедури **Visual Basic** перевіряє, чи потрібні даній процедурі

параметри. Переконавшись в тому, що потрібні (параметр message) він передає в процедуру рядок “Наша перша процедура”. Тобто фактично в процедурі відбувається присвоєння змінній **message** значення “Наша перша процедура”. Ну а далі відбувається виклик функції **MsgBox** і виведення повідомлення на екран. Якщо кількість параметрів, переданих при виклику процедури не співпаде з кількістю параметрів в оголошенні процедури - Visual Basic згенерує помилку.

Отже, із процедурами розібралися. Прийшов час розібратися з функціями.

Функції

Функція - це певний блок коду, який буде повертати значення. Цим, і тільки цим функції відрізняються від процедур. Загальний синтаксис функції:

```
[Public | Private | Friend] [Static] Function ім'я функції
[(arglist)] [As type]
    [тут певний код]
    [ім'я функції = вираз]
    [Exit Function]
    [тут теж може бути певний код]
    [ім'я функції = вираз]
End Function
```

Що значить “буде повертати значення”? Розглянемо функцію з уроку 8:

```
Public Function MyFunc() As Byte
    &nbsp;MyFunc = 234
End Function
```

```
c = MyFunc()
```

Коли ми говорили про вирар, ми говорили, що MyFunc - це вирар, зі значенням 234. Тобто тут, функція MyFunc повертає значення 234 (байт). Щоб задати це значення, необхідно привласнити імені функції вираз. У нашому випадку як вираз виступає число 234.

Давайте розглянемо більш практичний приклад. Напишемо функцію для обчислення квадрата числа. У функції буде 1 параметр типу **Integer** - число для зведення в квадрат. Функція буде повертати значення квадрата параметра. Тип значення, що повертається - Long:

```
Public Function Square(number As Long) As Long
    &nbsp;Square = number * number
End Function
```

Викликати функцію можна так:

```
b = Square (5)
```

А можна так, використовуючи нашу процедуру для виведення повідомлення на екран:

```
ShowMessage Square (5)
```

А можна і так:

Square 5

В останньому випадку повернуте значення функцій іде в нікуди, але сама функція благополучно виконається. Зверніть увагу на відсутність дужок в останньому виклику. Дужки тут не обов'язкові, на відміну від двох попередніх викликів. Там дужки обов'язкові. (ну це і зрозуміло, як би Visual Basic без дужок догадалися, де параметри для ShowMessage, а де для Square).

На наступному уроці ми познайомимося з елементами керування і напишемо невелику програму.

Урок № 15. Зводимо все разом

В цьому уроці ми спробуємо написати нашу першу програму на Visual Basic - програму для рішення квадратних рівнянь. Можливо, що ця програма і не дуже корисна в господарстві, але вона добре вас ознайомить із принципами програмування на Visual Basic. Отже, починаємо. *Згадаємо з уроку № 5 основні етапи розробки додаток на Visual Basic:*

- продумати програму (продумати, що саме програма повинна робити, які саме задачі повинна вирішити, реалізувати їх подумки, продумати структуру даних, і т.д.).
- проектування інтерфейсу, тобто розміщення на формі потрібних керуючих елементів, кнопок, списків і т.п. Цей етап називається складанням кістяка програми.
- написання програмного коду, який поєднує розміщені на формі керуючі елементи, тобто "нароцування плоті на кістяк".
- наладка програми. Цей етап часто займає більше часу, чим попередні.
- остаточна компіляція і, якщо це необхідно, створення дистрибутива (тобто інсталяційного файлу setup.exe).

Програму будемо писати згідно з цими пунктами:

1. Продумати програми.

Що повинна робити наша програма? - вирішувати квадратні рівняння. Згадаємо, як вирішуються квадратні рівняння.

$$a*x*x + b*x + c = 0$$

Щоб вирішити таке рівняння, потрібно знайти його дискримінант а потім, корені. Дискрименант шукається по формулі:

$$D = b*b - 4*a*c$$

А корені:

Якщо дискримінант > 0 , то

$$X1 = (b + (\text{корінь з } D)) / 2*a$$

$$X2 = (b - (\text{корінь з } D)) / 2*a$$

Якщо дискримінант $= 0$, то

$$X1 = X2 = b / 2 * a$$

Якщо дискримінант < 0 , то коренів не існує.

Отже, що вийшло:

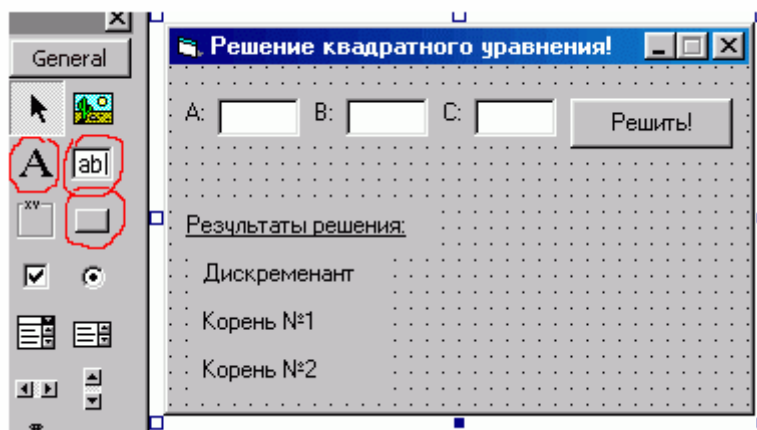
Вхідні дані в програму - коефіцієнти a,b,c. Дані будемо вводити в текстові поля для введення (TextBox).

Вихідні дані - корені (якщо вони є) і дискримінант. Дані будемо виводити в елемент мітку (Label). Запустити рішення будемо натисканням на кнопку (Command Button).

З першим пунктом розібралися. Переходимо до другого - проектування інтерфейсу майбутньої програми.

2. Проектування інтерфейсу.

Запустите Visual Basic. Виберіть тип Standart EXE. Помістіть на форму елементи керування, як показано на малюнку:



Потрібні елементи керування виділені червоним кольором. Для чого потрібні ці елементи? Елемент мітка (Label) потрібний для відображення тексту. Тобто служить як би підказкою. У ньому ми будемо виводити наш дискримінант і корені. Елемент текстове поле (TextBox) призначений для введення користувачем інформації. Введену інформації можна легко одержати в кодї програми. Для цього в TextBox є властивість Text. Ну, і нарешті, елемент кнопка (CommandButton), потрібний для запуску певної дії. У нашому випадку дією буде рішення квадратного рівняння. А запуститися вона буде при натисканні на кнопку.

Розмістити елементи керування краще в такий спосіб: помістіть спочатку на форму елемент **Label**. У вікні **Properties** змініть властивість **Caption** цього елемента на "A:". Потім перетягніть на форму елемент **TextBox** так, щоб він виявився поруч з елементом **Label**. Видаліть текст з елемента **TextBox** (властивість **Text**). Потім виділіть обидва елементи (використовуючи Ctrl) і скопіюйте їх у буфер обміну клавішами Ctrl+C. Далі натисніть Ctrl+V. **Visual Basic** запитає Вас, чи потрібно створити масив з елементів керування, що вставляються? Відповісти **Ні**. Тепер на формі з'явилася копія елементів **Label** і **TextBox** (для параметра **B**). Скопіюйте в такий же спосіб елементи для параметра **C**. У кнопки змініть властивість **Caption** на "Вирішити!". Змінюючи властивість **Caption** кнопки, Ви тим самим

змінюєте текст на самій кнопці. (Якщо цього не зробити, то користувачеві нашої програми важко було б здогадатися, що ж відбудеться при натисканні на кнопку Command1!). *Зверніть увагу на кнопки керування формою* (праворуч угорі). Їх 3 штуки: згорнути, розгорнути і закрити. Кнопка розгорнути нашій формі зовсім ні до чого. Давайте її заберемо. Для цього виділіть форму (клацніть по ній) і у вікні **Properties** відредагуйте властивість **BorderStyle**. Вам запропонують вибрати кілька значень зі списку, а саме:

0-None - у форми не буде заголовок і кнопок керування.

1-Fixed Single - форма буде мати заголовок і одну кнопку - закрити.

2-Sizeable - задається за замовчуванням. Форма має всі 3 кнопки.

3-Fixed Dialog - те ж, що і 1, але форму не буде видно на панелі задач

4-Fixed ToolWindow - скорочений вид заголовка й одна кнопка - закрити

5-Sizeable ToolWindow - те ж що і 4, але можна змінювати розміри форми

Вкажіть 1 номер (Fixed Single). Зверніть увагу на кнопки керування - залишилася тільки одна - закрити форму. Але куди ж ділася кнопка згорнути? Вона могла б стати нам в нагоді! Давайте повернемо її. Для цього потрібно встановити властивість форми **MinButton** у **True**. Тепер все ОК.

Ну і останній штрих. Зробимо текст “Результати рішення” підкресленим. Для цього виділіть **Label** і знайдіть у вікні **Properties** властивість **Font**. Якщо виділити цю властивість, то праворуч з’явиться маленька кнопочка, при натисканні на яку відкриється вікно вибору шрифту. Можливо це вікно ви вже десь бачили. Наприклад при виборі шрифту в Блокноті. Справа в тому, що це вікно - одне з загальних діалогів Windows (Common Dialogs). До них відносяться такі вікна, як “Відкриття файлу”, “Збереження файлу”, “Настроювання принтера”, “Вибір кольору”, “Вибір шрифту”, “Вибір каталогу”, “Пошук і заміна тексту”. такі вікна не потрібно створювати самому. Їх надає Windows. Отже, у вікні вибору шрифту поставте галочку - **Підкреслений**. Натисніть ОК.

Ви вже напевно давно помітили той факт, що в різних елементах керування деякі властивості часто повторюються (про це я говорив на попередніх уроках). Наприклад властивість **Caption**, **Name** або властивість **Font**. Більш того, властивість **Name** є в БУДЬ-ЯКОГО елемента керування. Це пов’язано з об’єктно-орієнтованим програмуванням (ООП). Ми не будемо докладно на цьому зупинятися, тому що зараз це не настільки важливо. Пізніше, я постараюся торкнутися цієї теми і розповісти про ООП. А поки прийміть це як факт.

Зверніть увагу на імена (властивість Name у вікні Properties) тільки що створених елементів. Label1, Label2... Text1, Text2.... Command1. При створенні нового елемента Visual Basic завжди дає такі імена сам. У наслідку вони можуть бути змінені. Узагалі, імена елементів керування дуже важлива річ. Такі ж важлива, як імена **змінних**. По цих іменах ви будете звертатися до ваших елементів керування в коді програми. Існують деякі рекомендації з іменування елементів керування. Завжди дотримуйте їх!

Давайте перейменуємо елементи керування. Для цього змініть властивість **Name** кожного елемента керування відповідно до таблиці:

Призначення елемента управління	Ім’я (властивість Name)
A: - параметр A	txtParamA
B: - параметр B	txtParamB
C: - параметр C	txtParamC
Кнопка для запуску рішення	cmdCalculate
Label, з обсиленим дискримінантом	lblD
Label, з коренем X1	lblX1
Label, з коренем X2	lblX2
Форма, яка містить всі ці елементи	frmMain

Інші елементи перейменовувати не обов'язково, тому що ми до них не будемо звертатися в кодї програми. Але, якщо хочете, можете перейменувати. Привчайтеся до цього.

Інтерфейс готовий. Він звичайно не претендує на звання самого зручного і красивого інтерфейсу року ;), - але для початку зійде. Все одно, згодом, уже після написання коду програми, Ви зможете змінити інтерфейс за Вашим смаком.

3. Написання програмного коду.

Тепер саме цікаве! Ми будемо писати код для нашої програми! Давайте ще раз продумаємо алгоритм роботи програми:

Задаємо вихідні дані в тектові поля (a,b,c). Нагадаю, що код для цього писати не потрібно. За нас все зробить **Visual Basic** і **Windows**. Ось в цьому і проявляється принадність графічного інтерфейсу користувача (**GUI**). Ми тільки рахуємо введені значення – от і все.

Після натискання на кнопку, робимо обчислення дискримінанта і коренів.

Виводимо отримані значення в мітку (Label'u).

Нам необхідно написати обробник події клік (**Click**) нашої кнопки - **cmdCalculate**. Що значить обробник події? *Обробник події* - це процедура, яка буде виконуватися кожний раз, коли відбудеться та або інша подія. Наприклад подія **Click**. Вона відбувається кожний раз при натисканні на кнопку. Тобто, якщо запустити програму на виконання і не натискати на кнопку **cmdCalculate** нічого не відбудеться. Але як тільки Ви натиснете кнопку, відбудеться виконання коду, який написаний в процедурі обробки події **Click** (процедурі з ім'ям **cmdCalculate_Click**). Код буде виконуватися щоразу, коли користувач натискатиме кнопку. Саме в цьому і є суть тої самої *подійно-керованої моделі програмування*, що відрізняється від плоских послідовних програм (Turbo Паскаля, наприклад). Програмування на **Visual Basic** цілковито базується саме на *подійно-керованої моделі програмування*.

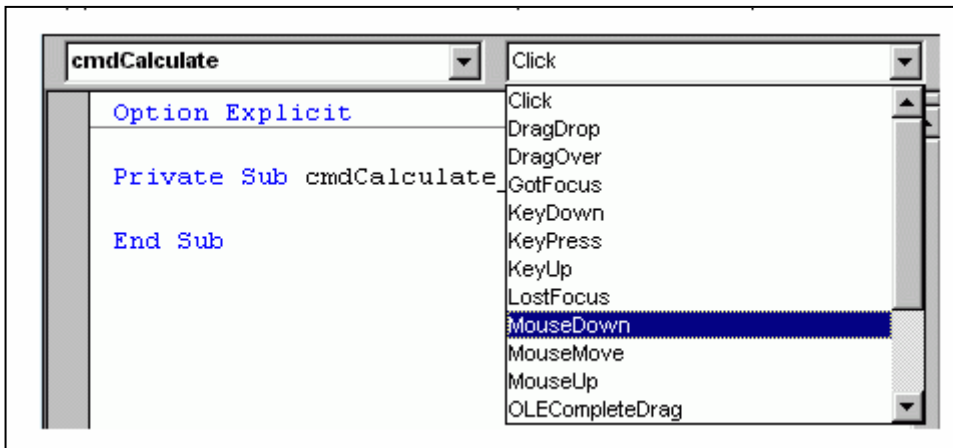
Сподіваюся, що Ви зрозуміли мою думку. Якщо ні, - то не панікуйте, далі все стане зрозумілим.

Щоб створити обробник події **Click** необхідно зробити подвійний клік по нашій кнопці **cmdCalculate** (при подвійному кліці **Visual Basic** створює заготовку обробника події – по стандарту. У кнопки, це подія **Click**, у форми - **Load**, у таймера - **Timer** і т.д.). *Visual Basic створить для вас заготовку процедури, котра буде виглядати в такий спосіб:*

```
Private Sub cmdCalculate_Click()
```

```
End Sub
```

Щоб створити обробник для іншої події, необхідно вибрати подію в правому списку у вікні коду:



У списку досить багато різних подій. У лівому списку Ви можете вибрати доступні елементи керування, які поміщені на форму. У даному випадку в списку ви бачите **cmdCalculate**. Дуже зручно.

Так само, як і певні властивості елементів керування, події також

повторюються. Наприклад, подія **Click**. Вона є і в елемента кнопки (**Command Button**), і в елемента мітки (**Label**) і в багатьох інших. Є вона й у форми. Це дуже полегшує процес програмування. Не потрібно витратити багато часу на вивчення кожного елемента керування, тому що багато чого повторюється.

Зауважте також, що в процедурі обробки події **Click** немає вхідних параметрів, про що нам говорять порожні дужки. Додати свої параметри в цю процедуру не можна. У процедур обробки інших подій можуть бути параметри. Наприклад, у події **MouseMove** (координати курсору миші) або **KeyUp** (код віджатої клавіші). Ці параметри передаються непомітно для програміста, і їх можна використовувати під свої потреби (а можна і зовсім не використовувати).

Тепер приступаємо безпосередньо до програмування. Для початку оголосимо змінні з типом **Double** (для збереження дійсних чисел):

```
Private Sub cmdCalculate_Click()
    ' оголошуємо змінні
    Dim paramA As Double
    &nbsp;Dim paramB As Double
    &nbsp;Dim paramC As Double

    &nbsp;Dim x1 As Double
    &nbsp;Dim x2 As Double
    &nbsp;Dim D As Double
End Sub
```

Тепер читаємо введені параметри **a**, **b** і **c**. Для цього привласнимо змінним **paramA**, **paramB** і **paramC** значення властивості **Text** усіх 3-х полів для введення (TextBox'ів). Ми можемо це зробити тому, що Visual Basic сам перетворить число у вигляді рядка в звичайне число з крапкою, що плаває.

```
paramA = txtParamA.Text
paramB = txtParamB.Text
param = txtParamC.Text
```

Доступ до властивості будь-якого елемента керування здійснюється через крапку, що розділяє ім'я властивості й ім'я елемента. Зверніть увагу на технологію **Intellisense**. Visual Basic видає список доступних властивостей цього елемента керування. Це дуже зручно. Вам не прийдеється вивчати напам'ять довгі і складні назви властивостей. Достатньо буде вибрати потрібну властивість зі списку.

Тепер нам потрібно розрахувати дискримінант. Для цього привласнимо змінній D, що буде зберігати значення дискримінанта, наступний вираз:

$$D = (\text{paramB} * \text{paramB}) - (4 * \text{paramA} * \text{paramC})$$

Дужки я поставив для наочності. Ставити їх у даному випадку не обов'язково, тому що множення все одно виконається до віднімання (згадайте, що множення має більший пріоритет, чим віднімання).

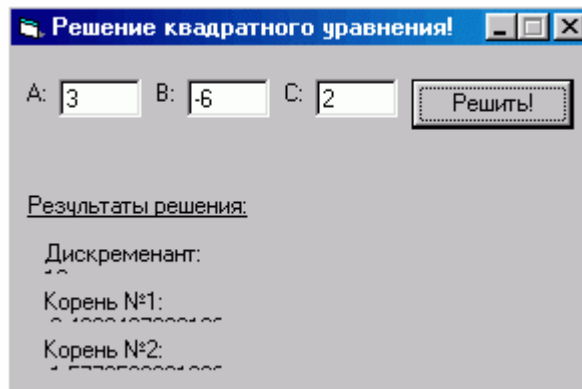
Тепер, знаючи значення дискримінанта, потрібно порівняти його з нулем. Якщо він більше нуля, то обчислити обидва корені, якщо дорівнює нулеві, то обчислити один корінь, ну а якщо менше, то не буде чого обчислювати і програма мусить повідомити про те, що коренів немає. Таке розгалуження ми організуємо за допомогою оператора If:

```
If D > 0 Then
    &nbsp;px1 = (paramB + Sqr(D)) / (2 * paramA)
    &nbsp;px2 = (paramB - Sqr(D)) / (2 * paramA)
    &nbsp;lbl.Caption = "Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корінь №1: " & x1
    &nbsp;lbl2.Caption = "Корінь №2: " & x2
ElseIf D = 0 Then
    &nbsp;px1 = paramB / (2 * paramA)
    &nbsp;px2 = x1
    &nbsp;lbl.Caption = " Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корінь №1: " & x1
    &nbsp;lbl2.Caption = "Корінь №2 = Кореневі №1"
ElseIf D < 0 Then
    &nbsp;lbl.Caption = " Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корнів немає!"
    &nbsp;lbl2.Caption = ""
    &nbsp;MsgBox " Дискримінант: менше нуля! Коренів немає!", vbCritical
End If
```

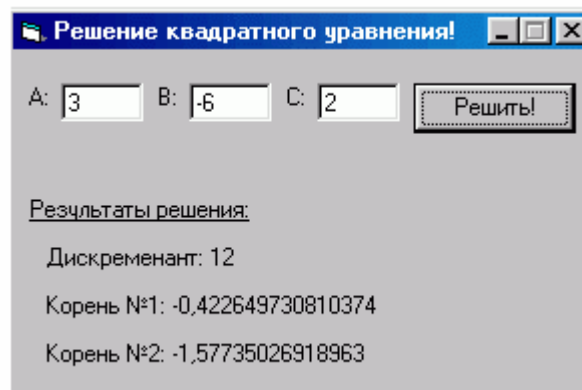
Не забувайте про відступи! З ними код є більш наочним.

Корінь ми обчислюємо вбудованою функцією VB - Sqr (від Square). У виразах, що ми привласнюємо властивості **Caption**, ми використовуємо оператор *конкатенації* (склеювання рядків). Ним ми склеюємо рядок ліворуч від & з рядком праворуч. Відкіля береться рядок праворуч? Адже там змінні типу Double!? Як я вже говорив, Visual Basic неявно займається перетворенням типів. У даному випадку перед конкатенацією, числа Double спочатку перетворюються в рядок.

У принципі, програма вже готова. Давайте перевіримо її працездатність. Натисніть кнопку Start. З'явиться наша форма. Введіть значення в поля: a = 3, b = -6, c = 2. Натисніть на кнопку "Вирішити!". *Якщо Ви все робили правильно, то повинні побачити наступну картинку:*



Справа в тому, що рядок для виведення в **Label** не вписується в його розміри. Тому відбувається перенос на наступний рядок. Щоб цього уникнути, закрийте програму й встановіть властивість міток **AutoSize** у True (Порада: Щоб не встановлювати цю властивість тричі для кожної мітки, виділіть їх відразу всі 3 і встановіть властивість). Властивість **AutoSize** - підганяє розмір мітки так, щоб текст у властивості **Caption** цілком умістився в один рядок і не переносився на інший. Тепер знову запусіть програму, введіть ті ж значення і натисніть на кнопку:

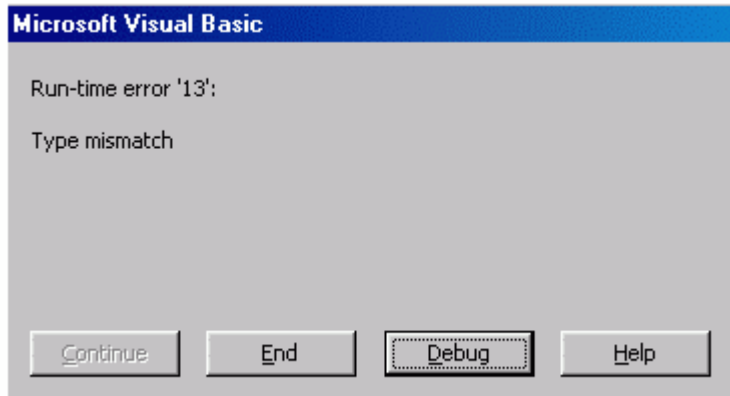


Тепер все в порядку! Програма працює! Можете поекпериментувати, вводючи різні значення коефіцієнтів.

Але, зверніть увагу! Що відбудеться, якщо ми, не ввівши значення в поля, натиснемо на кнопку? Що тоді присвоїться нашим змінним? Або, що буде, якщо ми введемо нулі як коефіцієнти? В обох випадках **Visual Basic** згенерує помилку. Чому? Відповідь на це питання ми розберемо на наступному уроці.

Урок № 16. Налаштування програми

Отже, на попередньому уроці ми знайшли *баги* (від слова Bug - жук) в нашій програмі, тобто недоліки (помилки). Від цих багів потрібно позбутися. Подивимося причину виникнення помилок. Запустіть програму. Нічого не прописуючи в поля натисніть на кнопку. Visual Basic видасть вікно, у якому скаже: "Type mismatch", тобто помилка в типах. У вікні доступні 3 кнопки:

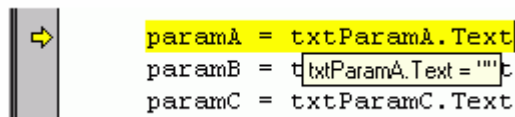


End - завершити додаток

Debug - показати місце виникнення помилки, щоб ми змогли від неї позбутися

Help - викликати довідку про помилку, яка виникла.

Натисніть Debug. Visual Basic покаже вам причину виникнення помилки:



Жовтим кольором виділений рядок - причина помилки. Якщо навести курсор миші на ім'я змінної, то спливе підказка, у якій Visual Basic повідомить нам її значення. Така можливість доступна лише в режимі **Debug**. Поточний режим можна довідатися із заголовка вікна Visual Basic. *Наприклад:*

у режимі проектування інтерфейсу це рядок:

Ім'я_Проекту - Microsoft Visual Basic [design]

при запусченому додатку:

Ім'я_Проекту - Microsoft Visual Basic [run]

у режимі Debug:

Ім'я_Проекту - Microsoft Visual Basic [break]

Наведіть мишкою на txtParam.Text. Visual Basic повідомить про те, що значення текстового поля txtParam.Text = "" (порожні лапки, або по іншому - порожній рядок). Тут можна здогадатися, чому відбувається помилка розбіжності типів. Адже змінна **param** повинна зберігати дійсне число, а тут ми намагаємося привласнити їй строкове значення - "". Якби **param** мала тип String, або в txtParam.Text був би рядок - число ("123") - помилка б не

виникла. А тут Visual Basic просто не може сам конвертувати рядок "" у число типу Double. Але ми повинні обов'язково знайти можливе рішення помилки. Давайте заглянемо в розділ *vbhelprus*. Знайдемо там функцію **Val**. ВАУ! Це саме те, що нам потрібно! *Давайте просто змінимо код:*

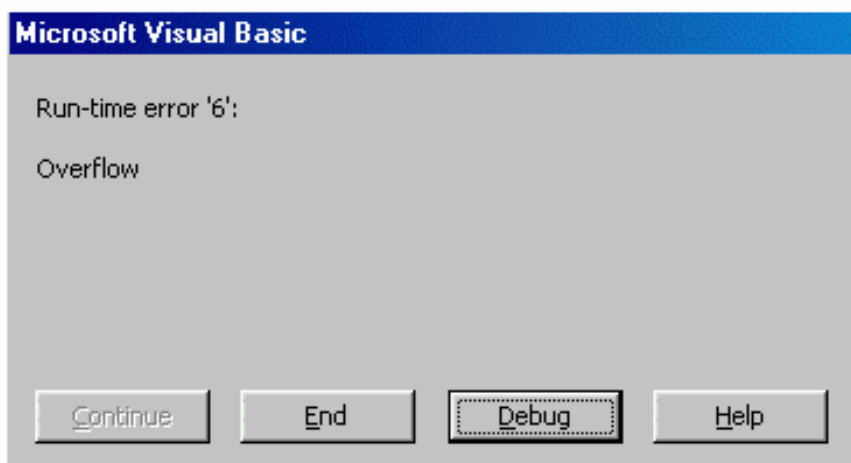
```
paramA = txtParamA.Text
paramB = txtParamB.Text
param = txtParamC.Text
```

На код:

```
paramA = Val(txtParamA.Text)
paramB = Val(txtParamB.Text)
paramC = Val(txtParamC.Text)
```

Тепер помилок розбіжності типів бути не може, оскільки функція **Val** при будь-якому параметрі поверне число, а сама функція **Val** ніколи не викликає помилки, хоча ... але це вже на совісті розробників VB :)

Проаналізуємо отриманий результат. Що змінилося після того, як ми усталили функцію Val? Тепер ми знаємо що змінним **paramA**, **paramB** і **paramC** у будь-якому випадку буде присвоєно число. Але от яке число? Адже якщо ми введемо в поля нулі, то і змінні будуть містити нулі! Також, якщо в поля ми введемо !букви!, то змінні також будуть містити нулі. Значить потрібно зробити перевірку на вміст у змінних нулів. Запустіть програму і введіть у поля нулі (або букви) і натисніть кнопку. Visual Basic видасть вікно:



Натискаємо Debug і от що бачимо:

```

ElseIf D = 0 Then
    x1 = paramB / (2 * paramA)
    x2 = x1
ElseIf D < 0 Then

```

paramA = 0

Причина помилка полягає в неможливості ділення на 0, а paramA у нас саме і дорівнює 0. До того ж при нульових коефіцієнтах квадратне рівняння вирішується набагато простіше (наприклад, якщо $c=0$, то x винесемо за дужку, ну а далі все просто). Позбудемося від цього

непорозуміння. Для цього, зробимо після присвоєння значення властивості Text змінним ще одну перевірку - перевірку на вміст нулів у **змінних**:

```
If paramA = 0 Or paramB = 0 Or paramC = 0 Then
    &nbsp;MsgBox "Нулі як коефіцієнти не допускаються!", _
    &nbsp;vbCritical
    &nbsp;Exit Sub
End If
```

Символ "_" використовується в тому випадку, коли ви хочете перенести частину виразу на інший рядок. У даному випадку ми переносимо константу **vbCritical**.

Тут ми перевіряємо змінні на вміст у них нулів. В принципі, можна було б перевірити не змінні, а самі текстові поля (If Val(txtParam.Text)=0 Then...). Це вже справа смаку. Все одно, результат однаковий.

Другий баг переможений! Можна запускати програму і вирішувати рівняння :).

Підсумковий код програми:

```
Private Sub cmdCalculate_Click()

' повідомляємо змінні
Dim paramA As Double
Dim paramB As Double
Dim paramC As Double

Dim x1 As Double
Dim x2 As Double
Dim D As Double

paramA = Val(txtParamA.Text)
paramB = Val(txtParamB.Text)
paramC = Val(txtParamC.Text)

If paramA = 0 Or paramB = 0 Or paramC = 0 Then
    &nbsp;MsgBox "Нулі як коефіцієнти не допускаються!", _
    &nbsp;vbCritical
    Exit Sub
End If

D = (paramB * paramB) - (4 * paramA * paramC)

If D > 0 Then
    &nbsp;x1 = (paramB + Sqr(D)) / (2 * paramA)
    &nbsp;x2 = (paramB - Sqr(D)) / (2 * paramA)
    &nbsp;lbl.Caption = "Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корінь №1: " & x1
    &nbsp;lbl2.Caption = "Корінь №2: " & x2
ElseIf D = 0 Then
    &nbsp;x1 = paramB / (2 * paramA)
    &nbsp;x2 = x1
```

```

&nbsp;lbl.Caption = "Дискримінант: " & D
&nbsp;lbl1.Caption = "Корінь №1: " & x1
&nbsp;lbl2.Caption = "Корінь №2 = Кореневі №1 "
ElseIf D < 0 Then
&nbsp;lbl.Caption = "Дискримінант: " & D
&nbsp;lbl1.Caption = "Корній немає!"
&nbsp;lbl2.Caption = ""
&nbsp;MsgBox "Дискримінант менше нуля! Корній немає!", vbCritical
End If

End Sub

```

Може бути ви вже помітили, що при кожному запуску програми, форма з'являється увесь час у різних місцях. Давайте позбудемося від цього, і зробимо так, щоб форма при кожному запуску з'являлася в центрі екрана. Для цього змінимо властивість `StartPosition` у `"2-CenterScreen"`.

Використання покрокового трасування:

Покрокове трасування - це метод налагодження додатка при якому можна виконувати код по одній команді і стежити за ходом її виконання. Це дуже корисний метод! Таким методом можна знаходити ті помилки, що не може знайти Visual Basic. *Такі помилки називаються логічними.* Тут я б хотів привести дуже цікавий і повчальний уривок із книги *"Програмування в середовищі Visual Basic 5"*:

“Перш ніж налагоджувати програму, необхідно переконатися в тому, що вона містить помилки. Тому тестування програми є першим кроком на шляху її налагодження. Одним зі способів тестування програми є її запуск із різними вихідними даними і спостереження за правильністю її функціонування. (чим ми вище і займалися). Цей процес закінчується в тому випадку, коли програма коректно реагує на всі дані, що вводяться. Однак цей спосіб не є ефективним для великих програм. У будь-якому випадку вибір тестових даних для перевірки коректності роботи є досить важливим. Вибір необхідних тестових даних пов'язаний з великими труднощами, тому що треба перевірити всі можливі шляхи виконання програми для того, щоб бути упевненим у відсутності помилок.”

“Тестування програм - це більшою мірою мистецтво, чим наука. Не існує яких-небудь правил, що завжди спрацьовують. Якщо програма має великий розмір коду або досить складну структуру і немає впевненості в тому, що вдасться перевірити її по всіх можливих шляхах виконання, то в цьому випадку Вам варто підійти до процесу тестування з погляду здорового глузду. Наприклад, фірма Microsoft не має можливості перевіряти кожен нову версію операційної системи (ОС) на всіх можливих конфігураціях апаратного забезпечення.”

“Ключ до успішного тестування лежить у словосполученні *“здоровий глузд”*, і в наступній історії буде показана життєва важливість цієї концепції. Одна комунальна компанія використовувала досить складну, але, як вважалося, надійну програму для надсилання рахунків на оплату і прийому квитанцій. У випадку коли компанія не одержувала оплаченої квитанції від користувача, той автоматично позбавлявся наданих йому комунальних послуг. У той день, коли відбулася ця історія, один із клієнтів компанії, ідучи у відпустку, відключив електроенергію. У результаті чого комп'ютер надіслав йому рахунок за електроенергію в розмірі \$0.00, яуйй, цілком природньо, не міг бути оплачений. Після визначеного числа запитів на оплату комп'ютер вирішив, що той клієнт, якому був надісланий рахунок, не сплатив вчасно ці самі \$0.00, - і як наслідок він залишився без електрики!”

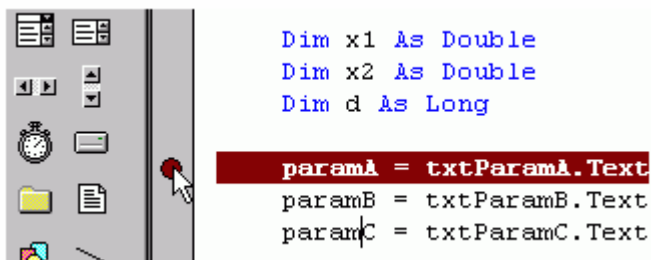
“В історії не говориться про реакцію розгніваного клієнта, однак, якщо ця історія відбулася насправді, - то провина програміста, який написав додаток, у тому, що він просто забув вставити перевірку на наявність порожніх рахунків. Цілком очевидно, що він просто не припускав, що подібна ситуація може виникнути. (Хоча, по правді кажучи, багато програмістів використовують в операторах порівняння виразу $a > b$, замість $a >= b$).”

“Мораль цієї історії така, що потрібно для перевірки коректності роботи програми використовувати граничні значення, тому що саме такі значення дозволяють знаходити помилки.”

Ось так, шановні читачі. Як бачите програм без помилок не буває. Тепер я думаю вам зрозуміло, чому час від часу приходиться робити переінсталяцію Windows :))).

Ну а тепер повернемося до відлажування (або як жартують програмісти – до рятування програми від лажі ;). На чому ми зупинилися? Ах да, - на покроковому трасуванні і логічних помилках. От саме така логічна помилка залишила клієнта без електроенергії. *Для знаходження таких помилок дуже зручно використовувати покрокове трасування.*

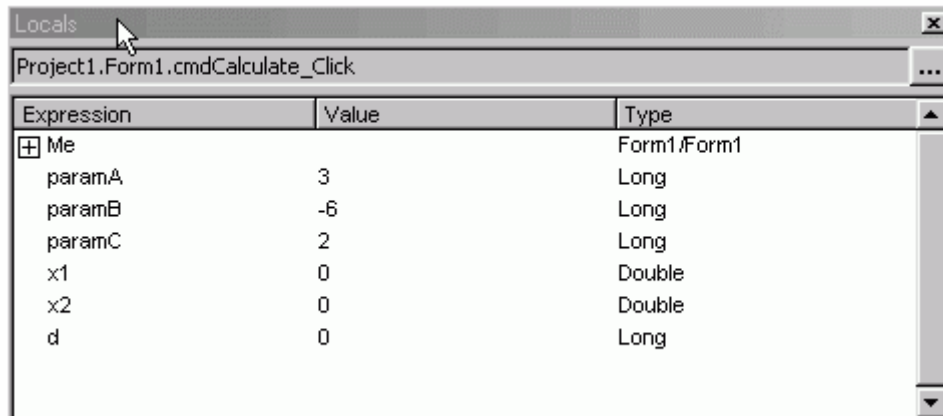
Як використовувати таке трасування? Потрібно всього навсього в потрібному місці програми поставити крапку останова (брейкпоінт Breakpoint). Дійшовши до такого місця, Visual Basic призупинить виконання програми і переведе вас у режим **Debug**, де Ви зможете виконувати код покомандно. Давайте поставимо брейкпоінт у нашій програмі, і простежимо хід її виконання. Щоб поставити крапку останова потрібно клікнути на вертикалі ліворуч від коду:



А можна і по іншому. Натисніть правою кнопкою миші на тому рядку коду, де Ви хочете поставити брейкпоінт і в меню вибрати Toggle->Breakpoint. *Зауваження:* таку крапку не можна ставити на рядку з оголошенням змінної.

Поставте крапку як показано на малюнку і запустіть програму. Введіть наступні дані в якості параметрів: $a = 3$, $b = -6$, $c = 2$. Натисніть на кнопку "Вирішити!". Visual Basic зупинить виконання програми точно на тому рядку, на якому стоїть брейкпоінт. Тепер Visual Basic у режимі **Debug**. Ви можете переглядати значення **змінних**, наводячи на них курсор миші і виконувати програму по кроках. Щоб виконати одну команду, потрібно натиснути F8 або Shift+F8. Відмінність цих двох команд полягає в тому, що при натисканні F8 на рядку з викликом процедури Visual Basic ввійде в цю процедуру і Ви зможете продовжити покрокове виконання в цій процедурі до рядка End Sub, де VB поверне вас на те місце, де відбувся виклик. Якщо ж натиснути Shift+F8, то Visual Basic виконає процедуру, але не буде заходити в неї для покрокового трасування.

Тепер протрасуйте програму. Простежте за тим, як змінюються значення **змінних** при присвоєнні їм значення. Тепер відкрийте вікно **Locals Window** (меню View->Locals Window). У цьому вікні відображаються значення всіх локальних **змінних**:



Expression	Value	Type
Me		Form1/Form1
paramA	3	Long
paramB	-6	Long
paramC	2	Long
x1	0	Double
x2	0	Double
d	0	Long

Тут зображені всі наші локальні змінні. Їх 6 штук. “А от що ж таке Me?” - запитаете Ви. Скоріше за все операційна система Windows Millenium? :) А от і ні, не вгадали, “Me” - це також зарезервоване слово Visual Basic. Воно вказує на поточну форму. У своїй програмі Ви можете звертатися до своєї форми через Me. Наприклад, якщо написати

```
Me.Hide
```

то з екрана сховається та форма, у кодї якої написаний цей код (вибачте за каламбур).

Також можна помітити значок “+” ліворуч від “Me”. При натисканні на нього відбудеться розкриття списку з усіма дочірніми об’єктами форми. Там же будуть знаходитися всі елементи керування, які поміщені на форму, а також усі глобальні змінні рівня цієї форми.

Зауважте, що вікно **Locals** - плаваюче, і при подвійному щиглику мишкою по заголовку вікна воно “пристикується” у середовище VB. При черговому подвійному щиглику вікно “відстичується”.

Дотрасуйте (тобто пройдіть пошагово) програму до кінця. Закрийте її.

Обробка інших помилок

Незалежно від того, наскільки якісно написаний додаток, ніколи не можна цілком виключати можливість виникнення помилки в програмі. Ви напевно бачили наступне віконце Windows: “Програма виконала неприпустиму операцію і буде закрита...”. Тобто виникла виняткова ситуація, і Windows, не знаючи як можна обійти помилку, видає “загальне” вікно для всіх отаких помилок. Отож і у Ваших програмах також може виникнути така виняткова ситуація. Причин для цього може бути дуже багато. Наприклад, у користувача програмою збої з вінчестером, або глючить операційна система, або вірус видалив потрібний вам файл і т.д. і т.п. Загалом, у таких ситуаціях бажано повідомити користувачеві про те, що виникла помилка і необхідно просто спокійно вийти з програми а потім продовжити програму. *Про те як це зробити і буде розказано далі.*

У Visual Basic існують кілька глобальних об'єктів. Це **Err**, **Printer**, **Screen**, **Clipboard** і **App**. Ці об'єкти доступні в будь-якому місці програми. У даному випадку нас цікавить тільки об'єкт **Err**. Про призначення і використання інших об'єктів, можливо, розповім у статтях.

Об'єкт **Err** містить у собі 6 властивостей і 2 методи. Цей об'єкт містить інформацію про останню помилку, що відбулася. От найбільш важливі властивості:

Err.Number
Err.LastDllError

Ці властивості містять номер останньої помилки, яка відбулась. *Перша* - містить номер помилки самого Visual Basic. *Друга* - номер помилки при роботі з API функціями. Поки що нам знадобиться тільки перша властивість.

Visual Basic має у своєму розпорядженні оператор, за допомогою якого можна контролювати хід програми при виникненні помилок. Це оператор **On Error**. Він має кілька видів:

On Error GoTo МІТКА
On Error Resume Next
On Error GoTo 0

Перший оператор дозволяє вказати Visual Basic мітку (номер рядка) на яку буде передаватися керування програми при виникненні помилки. Розглянемо приклад використання цього оператора:

```
Private Sub Command1_Click()
    Dim myString As String
    &nbsp;On Error GoTo ERRH
    &nbsp;myString = "ERROR HANDLING WITH VB IS COOL"
    &nbsp;MsgBox Mid(myString, 0, 1) '<-i- тут помилка (#)
    &nbsp;Exit Sub '<-i- достроково виходимо, якщо немає помилок
ERRH: '<-i- мітка
    &nbsp;MsgBox Error(Err.Number) '<-i- виводимо свою помилку
End Sub
```

Що тут відбудеться при виконанні цього коду? Отже, спочатку з'являється змінна myString. Потім ми використовуємо оператор On Error.... У якості мітки вказуємо мітку ERRH (Мітка в VB - це слово з двокрапкою на кінці). Тобто тут ми вказуємо VB, що при виникненні помилки потрібно передати керування на мітку ERRH. Далі ми привласнюємо рядкові деяке значення. Поки що все в рамках правил. А от у рядку (#) відбувається помилка, а саме - функція **Mid** видає помилку *“Invalid procedure call or argument”* (Невірний виклик процедури або аргумент). Це відбувається, тому що другий параметр функції **Mid** менше одиниці. А він повинен бути більшим або дорівнювати їй (так написано в документації). Таким чином, відбувається помилка з номером 5. У нашому обробнику помилки ми виводимо **MsgBox** з описом помилки, яка виникла. Функція **Error** має один аргумент - номер помилки. Функція повертає рядок - опис помилки. Тепер запустіть цей код (Для цього помістіть на форму кнопку і вставте цей фрагмент у код форми). Ви побачите наш **MsgBox** с помилкою. Зверніть увагу, що цей **MsgBox** видаємо ми, а не сам Visual Basic, як було б, якби не було оператора On Error. Також зверніть увагу на оператор **Exit Sub**, який ми поставили перед міткою **ERRH**. Якщо Ви пам'ятаєте, цей оператор призначений для дострокового виходу з процедури. Він тут необхідний для того, щоб випадково не виконався наш обробник

помилки в тому випадку, коли помилки немає. Тобто, якщо Ви зміните другий аргумент функції Mid з 0 на 1, забравши в такий спосіб причину помилки, то Ви побачите тільки MsgBox у рядку (#). Код після ERRH виконуватися не буде.

Отже, висновок. Використовуйте оператор **On Error** тоді, коли Ви не цілком упевнені в правильності виконання того або іншого коду. Наприклад, Ви можете перед відкриттям файлу поставити On Error із вказівкою мітки, де Ви зможете спокійно обробити випадок, коли, наприклад, файлу немає на диску, або диск не доступний і т.д.

Необхідно підкреслити той факт, що тепер при виникненні помилки Visual Basic не видає своє вікно з описом помилки і нашою улюбленою кнопкою Debug :). Це можна виправити. Для цього натисніть правою кнопкою в будь-якому місці коду програми і виберіть Toggle->Break On All Errors. Тепер Visual Basic завжди буде переводити програму в режим Debug при виникненні будь-якої помилки (цілком природньо, що це не стосується логічних помилок, на них не діє жоден оператор :).

Давайте тепер подивимося на оператор **Оператор On Error Resume Next**. Його можна використовувати тоді, коли Ви просто хочете придушити помилку. Тобто продовжити програму без усяких попереджень. Замініте в нашому прикладі On Error GoTo ERRH на On Error Resume Next і подивіться що відбудеться. Ви не побачите ніяких ознак помилки. Код просто буде продовжений з наступного рядка. У нашому випадку - це Exit Sub.

Оператор On Error GoTo 0 видаляє всі раніше встановлені обробники помилок, тобто дозволяє Visual Basic використовувати його власний обробник помилок (При виникненні якої, він закриє додаток. А це іноді зовсім ні до чого).

За більш докладною інформацією можна буде звернутися до довідників.

Переходимо до наступного уроку.

Урок № 17. Доводимо до розуму

На цьому уроці ми навчимося оформляти програму у вигляді функцій і процедур. Це дуже важливо. Принцип модульного програмування дуже полегшує програмування і налагодження. *Модульне програмування* - означає поділ коду програми на окремі фрагменти, кожний з яких виконує чітко визначену задачу. Це особливо важливо для складних програм.

Доведемо до розуму нашу програму, для обчислення коренів квадратного рівняння. Як? Дуже просто! Давайте оформимо деякі частини коду у вигляді процедур і функцій. Наприклад, напишемо функцію для обчислення дискримінанта. І напишемо процедуру, яка буде виводити отримані значення в мітки. Чому саме процедуру? Тому що процедури пишуться для виконання певної послідовності дій, де не потрібно повертати яке-небудь значення. Давайте ще раз подивимося на ту частину коду, де відбувається перевірка значення дискримінанта й обчислення коренів рівняння:

```
If D > 0 Then
    &nbsp;px1 = (paramB + Sqr(D)) / (2 * paramA)
    &nbsp;px2 = (paramB - Sqr(D)) / (2 * paramA)
    &nbsp;lbl.Caption = "Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корінь №1: " & x1 ' <- тут
    &nbsp;lbl2.Caption = "Корінь №2: " & x2
```



```

ElseIf D = 0 Then
    &nbsp;px1 = paramB / (2 * paramA)
    &nbsp;px2 = x1
    &nbsp;lbl.Caption = "Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корінь №1: " & x1 ' <- і тут
    &nbsp;lbl2.Caption = "Корінь №2 = Кореневі №1"
ElseIf D < 0 Then
    &nbsp;lbl.Caption = "Дискримінант: " & D
    &nbsp;lbl1.Caption = "Корній немає!" ' <- і тут, теж
    &nbsp;lbl2.Caption = ""
    &nbsp;MsgBox "Дискримінант менше нуля! Корній немає!", vbCritical
End If

```

Зауважте, що в кожній з гілок оператора **If** спостерігається присвоєння властивості `Caption` 3-х міток (`lbl`, `lbl1`, `lbl2`). Тому цілком логічно, що цю ділянку можна оформити у вигляді процедури, параметрами якої будуть значення для міток.

Ну, що ж, починаємо?

Спочатку напишемо функцію для обчислення дискримінанта. Щоб обчислити дискримінант необхідно знати 3 параметри - коефіцієнти a , b і c . Функція буде повертати значення - дискримінант. Щоб самому не писати заготовку для цієї функції, Visual Basic надає можливість для автоматичного створення такої заготовки. Для цього виберіть у головному меню VB: Tools->Add Procedure... З'явиться вікно, у якому Вас попросять вказати вид процедури, що додається. Відзначте радіокнопку **Function**. У полі "Name:", вкажіть ім'я функції, що додається. Давайте назвемо її, наприклад, як **CalcDiscremenant**. Не бійтеся давати довгі назви процедурам і функціям! Все рівно, коли Ви захочете використовувати цю процедуру в коді, вам не прийдеться повністю набивати на клавіатурі її ім'я. Але про це трохи нижче.

Отже, натисніть OK і Visual Basic створить для вас наступну заготовку:

```

Public Function CalcDiscremenant()

End Function

```

Тепер модифікуємо її так, щоб функція приймала 3 аргументи (параметра) типу `Double`, і повертала значення, що теж буде мати тип `Double`:

```

Public Function CalcDiscremenant(a As Double, _
b As Double, c As Double) As Double

End Function

```

Залишилося додати код у нашу заготовку. Зробити це не важко:

```

Public Function CalcDiscremenant(a As Double, _
b As Double, c As Double) As Double
    &nbsp;CalcDiscremenant = (b * b) - (4 * a * c)
End Function

```

Необхідно підкреслити, що в Visual Basic параметри можна передавати двома способами: за значенням і по посиланню. Перший спосіб передає в процедуру (або функцію) тільки значення переданої змінної. У середині функції змінити значення такої змінної буде не можна. Другий спосіб передає у функцію посилання на передану змінну, і її значення легко можна бути змінити в тілі функції. За замовчуванням Visual Basic завжди передає параметри по посиланню. У нашому випадку параметри **a**, **b** і **c** не змінюються усередині функції. Тому можна зробити так, щоб параметри передавалися за значенням. Для цього перед ім'ям змінної необхідно поставити ключове слово ByVal (By Value), от так:

```
Public Function CalcDiscremenant(ByVal a As Double, _
ByVal b As Double, ByVal c As Double) As Double
```

У принципі, робити це не обов'язково, але все-таки бажано. Це підвищує наочність. Тепер, кожної, хто скористається нашою функцією відразу буде знати, що значення переданих **змінних** усередині функції мінятися не будуть (та й не можуть).

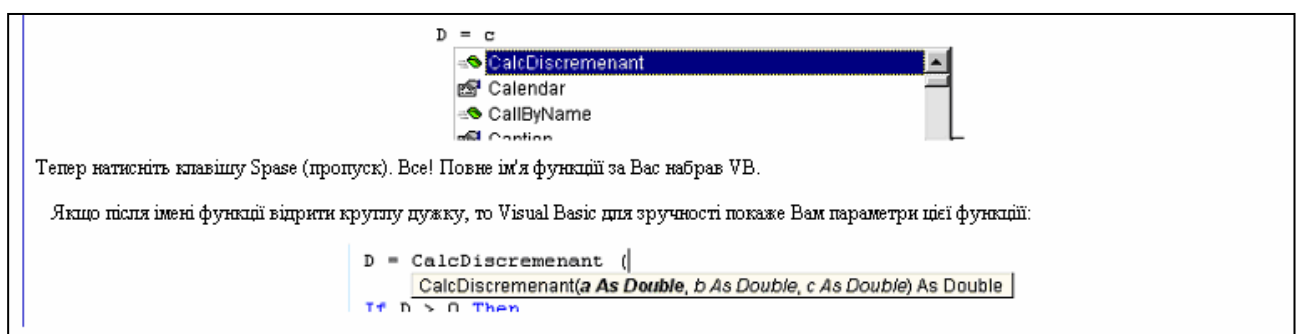
Тепер у нас є функція, що буде приймати 3 коефіцієнти **a**, **b**, **c** і обчислювати дискримінант. Давайте випробуємо її в дії. Для цього замініть в нашій програмі рядок, де обчислювався дискримінант:

$$D = (\text{paramB} * \text{paramB}) - (4 * \text{paramA} * \text{paramC})$$

на рядок:

$$D = \text{CalcDiscremenant}(\text{paramA}, \text{paramB}, \text{paramC})$$

Тут читач може заперечити – “Як же? Ви ж говорили, що ім'я функції не прийдеться набирати на клавіатурі!”. Так воно і є! Найшвидший спосіб замінити старий рядок на новий так: (не користуючись Ctrl+C, Ctrl+V, звичайно :) виділіть все, що знаходиться після знака “=”. Видаліть клавішею Delete. Натисніть Ctrl+J. Visual Basic запропонує Вам вікно, у якому знаходяться всі доступні на даний момент функції, константи, властивості, перерахування... і т.п., включаючи користувальницькі, тобто наші ;) . Натисніть на клавіатурі всього одну букву “c”, і Ви побачите нашу, тільки що створену функцію!:



Це дуже зручно. Не потрібно завжди пам'ятати всі параметри всіх функцій. Досить написати ім'я функції і поставити круглу дужку.

Примітка: Якщо у вас пропала така спливаюча підказка (наприклад, Ви клацнули в іншому місці коду програма), то повернути її можна, знову поставивши відкриваючу круглу дужку після імені функції. А можна ще простіше - натиснути комбінацію клавіш Ctrl + Shift + I.

Як бачите, програмувати на Visual Basic дуже просто. Причому, ця простота дуже добре поєднується з багатими можливостями самої мови. У результаті чого, на VB можна легко написати навіть дуже могутню програму. Звичайно, нашу програму могутньою не назвеш, адже це тільки маленький приклад можливостей VB! Дійте! Все у Ваших руках!

Тепер можете перевірити працездатність нашої функції. Як бачите, усе прекрасно працює.

Залишилося написати процедуру для виведення результатів у мітки. Додамо заготовку так само, як і в попередньому випадку (Tools->Add Procedure). Тільки цього разу не будемо відзначати радіокнопку Function, а залишимо все як є (Sub). Як ім'я процедури можна ввести, ну, скажімо WriteResultsInLabels. Чим довша назва, тим зрозуміліше, чим займається дана процедура або функція. Натиснемо ОК:

```
Public Sub WriteResultsInLabels()
```

```
End Sub
```

Зробимо так, щоб наша процедура приймала 3 параметри - значення, які необхідно вивести у відповідні мітки:

```
Public Sub WriteResultsInLabels (lblDCapt As String, _
lblX1Capt As String, lblX2Capt As String)
```

```
End Sub
```

Вхідні параметри мають тип String, тому що передавати в процедуру ми будемо саме рядки.

Для того, щоб процедура працювала, необхідно додати в неї код:

```
Public Sub WriteResultsInLabels (lblDCapt As String, _
lblX1Capt As String, lblX2Capt As String)
    &nbsp;lblD.Caption = lblDCapt
    &nbsp;lblX1.Caption = lblX1Capt
    &nbsp;lblX2.Caption = lblX2Capt
End Sub
```

Готово. Можна використовувати процедуру. Давайте замінімо кожні 3 рядка, де ми виводили значення в мітки. Вийде щось на зразок:

```
If D > 0 Then
    &nbsp;x1 = (paramB + Sqr(D)) / (2 * paramA)
    &nbsp;x2 = (paramB - Sqr(D)) / (2 * paramA)
    &nbsp;WriteResultsInLabels "Дискримінант: " & D, _
    "Корінь №1: " & x1, "Корінь №2: " & x2
ElseIf D = 0 Then
    &nbsp;x1 = paramB / (2 * paramA)
    &nbsp;x2 = x1
    &nbsp;WriteResultsInLabels "Дискримінант: " & D, _
    "Корінь №1: " & x1, "Корінь №2 = Кореневі №1"
ElseIf D < 0 Then
    &nbsp;WriteResultsInLabels "Дискримінант: " & D, _
    "Корній немає!", ""
```

```

        MsgBox "Дискримінант менше нуля! Корній немає!", vbCritical
End If

```

Програма готова. Звичайно можна було б ще оформити функцію, для перевірки правильності введених у поля (TextBox) значень. Функцію для обчислення коренів рівняння і т.д. Справа Ваша. Адже це був усього лише вступний курс. Я тільки хотів показати, як можна використовувати можливості Visual Basic на прикладі простої програмки - програмки для рішення квадратних рівнянь. Звичайно, багато чого залишилося за кадром. Наприклад, використання інших компонентів Visual Basic, таких як Timer, PictureBox і багатьох інших, а також додаткових ОСХ компонентів, використання API функцій Windows, створення Active компонентів, класів і т.д. і т.п. Усього і не перерахуєш. Цілком можливо, що згодом напишу і про це. А поки, переходимо до останнього уроку - компіляції додатка.

Урок № 18. Компіляція

Тепер наша програма налагоджена і готова до використання. Тепер наступив момент довідатися, як же відкомпілювати програму в ехе-файл?

Нагадаю, що Visual Basic пропонує 2 компілятора. Компіляція в *P-код*, і компіляція в *Native-код*. *P-код* - це старий компілятор і користуватися ним я не рекомендую. Завжди компілюйте додаток у *Native-код*. Вибір виду компіляції знаходиться на вкладці **Compile** у меню Project->Project Properties. Там також можна вказати кілька доступних видів оптимізації (про них розповім трохи нижче).

Отже, щоб відкомпілювати нашу програму, необхідно проробити наступні маніпуляції:

У меню **File** вибрати **Make ім'я_проекту.exe**

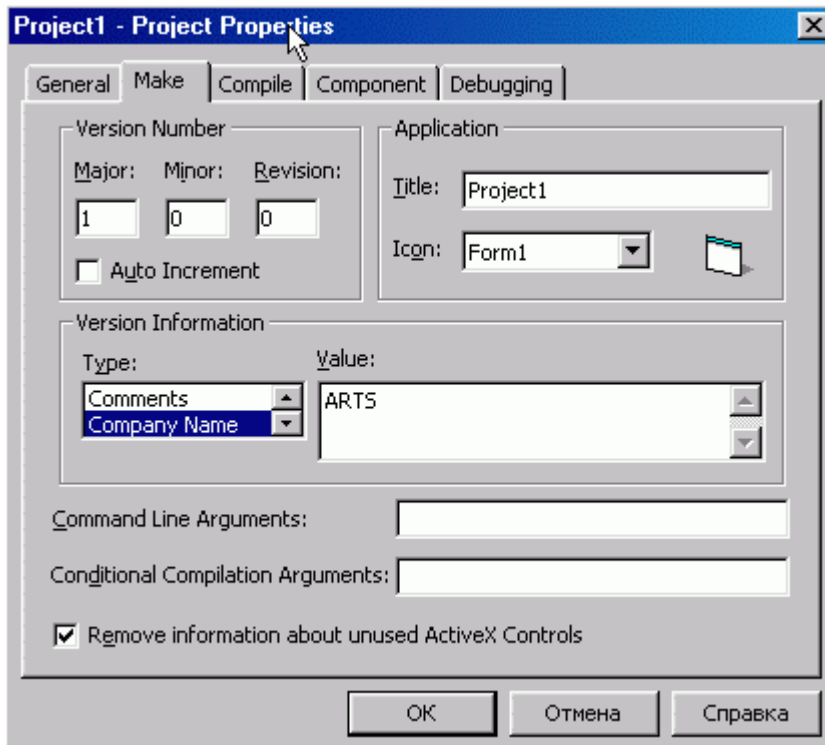
Ввести ім'я вихідного ехе-файлу

Якщо необхідно, то вибрати деякі опції, натиснувши на кнопку **Options**. (До цих опцій також можна добратися через меню Project->Project Properties).

Натиснути ОК. І, якщо Visual Basic не знайде ніяких помилок у програмі, то відкомпілює її і збереже в каталозі, який Ви вказали та під зазначеним Вами ім'ям.

Тепер можна запустити отриманий ехе-файл і перевірити його працездатність. Тепер Ви можете транспортувати його куди хочете, - викладати в інтернет, перемістити в певне місце і т.д. Єдине, що потрібно не забувати, так це те, що для запуску програми необхідна бібліотека msvbvmX0.dll, яка повинна знаходитися в каталозі Windows\System. X - версія Visual Basic. (5 або 6).

Тепер давайте подивимося на доступні опції додатка. Відкрийте вікно **Project Properties** (Project->Project Properties). Відкрийте вкладку **Make**:

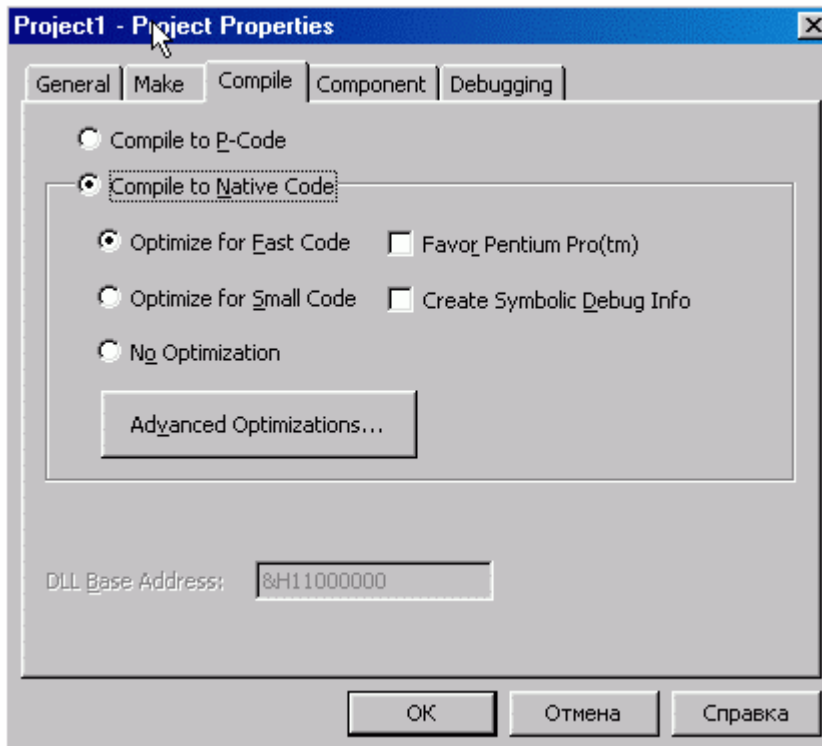


У рамці **Version Number** можна вказати версію додатка. Подивитися інформацію про версію додатка можна, вибравши вкладку “Версія” у властивостях exe-файлу. **Auto Increment** означає автоматичне збільшення версії **Revision** при кожній компіляції додатка (використовуючи **Make**).

У рамці **Application** можна вибрати заголовок додатка (під таким заголовком Ваш додаток буде видно в списку задач Window (Ctrl+Alt+Del)). Також можна вказати іконку для exe-файлу. У списку можна вибрати доступні форми в додатку. Іконку можна задати за допомогою властивості форми “*Icon*” (вас попросять вказати файл, з розширенням .ico або .cur).

У рамці **Version Information** можна вказати різну інформацію про програму. Подивитися цю інформацію можна буде, вибравши вкладку “*Версія*” у властивостях exe-файлу.

Тепер подивимося, що доступно на вкладці **Compile**:



Я рекомендую залишати все так, як є, - оскільки це є самий оптимальний варіант. Але якщо, з якихось причин Вас не влаштовує стандартний варіант, то можете змінити ці опції, керуючись наступною таблицею:

Опція	Опис
Optimize For Fast Code	Збільшує швидкодію скомпільованих виконавчих файлів, вказуючи компілятору, що швидкість важливіша розміру. Коли компілятор транслює оператори Visual Basic в машинний код, то досить часто в наявності є вибір серед багатьох різноманітних послідовностей машинного кода, які можуть правильно подати даний оператор або конструкцію. Іноколи ці розбіжності дозволяють досягти певних цілей компіляції. Вибір даної опції гарантує, що, коли компілятор розпізнає такі варіанти, він завжди буде генерувати саму швидку можливу послідовність кодів, навіть коли це може збільшувати розмір відкомпільованої програми.
Optimize For Small Code	Мінімізує розмір скомпільованих виконавчих файлів, вказуючи компілятору, що розмір важливіший за швидкодію. Вибір цієї опції гарантує, що, коли компілятор розпізнає варіанти послідовностей кодів, то він завжди буде генерувати найменшу можливу послідовність, навіть коли це може знизити швидкодію виконання скомпільованої програми.
No Optimize	Відключає будь-які оптимізації. Якщо вибрана дана опція, компілятор генерує код, який буде значно повільніший і більший по розміру, чим при виборі будь-якого типу оптимізації.
Favor Pentium Pro	Оптимізує генерацію об'єктного коду з урахуванням особливостей архітектури процесора Pentium Pro (P6). Код, згенерований з такою опцією, буде працювати також на більш ранніх процесорах, але менш

	ефективно. Деякі стратегії генерації об'єктного коду, які можна мпокійно застосовувати для Pentium Pro, не підтримуються на комп'ютерах з процесорами 80386, 80486 та Pentium. Як наслідок, використовувати зазначену опцію можна лише тоді, коли на всіх або на більшості машин, що виконують програму, використовується Pentium Pro.
Greate Symbolic Debug Info	Генерує в компільованому виконавчому файлі символічну інформацію наладки. Програми, компільовані у внутрішній код з використанням цієї опції, можуть бути налаштовані в Visual C++ 5.0 або іншим сумісним наладчиком. Установка цієї опції буде генерувати файл .pdb з необхідною інформацією наладки.

І опції у вікні Advanced Optimization:

Опція	Опис
Asume No Aliasing	Повідомляє компілятору, що програма не використовує суміщення імен (псевдоніми). <i>Псевдонім</i> – це ім'я, яке відноситься до розділу пам'яті вже адресованому іншим іменем. Псевдоніми використовуються при передачі параметрів ByVal, які посилаються на одну змінну. Використання цієї опції дозволяє компілятору реалізувати оптимізації, які інакше виконати неможливо, наприклад зберігання змінних в регістрах та оптимізація циклів. Однак це може обумовити неправильне виконання програми, - а тому, якщо параметри передаються по посиланню, варто використовувати зазначену опцію дуже обережно.
Remove Array Bounds Checks	Виключає контроль помилок допустимих індексів масиву та кількості розмірностей масиву. За замовчуванням Visual Basic проводить контроль при кожному зверненні до масива, визначаючи, чи знаходиться індекс всередині інтервала масива. Якщо індекс поза межами масива, то повертається помилка. Вибір цієї опції відключає контроль цієї помилки, що може значно прискорити маніпулюванням масивами. Однак, якщо програма звертається до масива по індексу, який не попадає в межі розмірності, без попереджень може здійснитись доступ до недостовірних розділів пам'яті. А це буде зумовлювати непередбачену поведінку або сбої програми.
Remove Integer-Overflow Checks	Відключає виведення помилок у випадках, коли числові значення, які привласнюються цілим змінним, розташовуються поза правильним інтервалом для типу даних. За замовчуванням Visual Basic виконує перевірку при кожному обчисленні із змінною цілочисленого типу даних (Byte, Integer, Long та Currency), аби гарантувати, що результуюче значення знаходиться всередині необхідного інтервала. Якщо значення має неправильну величину, - то видається помилка. Вибір даної опції відключає контроль даної помилки, що може прискорити цілочисленні обчислення. Однак у випадку переповнення ємності типу даного помилка не буде повернута, і як наслідок можуть мати місценоправильні результати.
Remove Floating-Point Error Checks	Відключає контроль помилок, який гарантує, що числові значення, які привласнюються змінним з плаваючою комою, знаходяться в правильному

	інтервалі для типів даних і що не буде відбуватись ділення на нуль та інші недопустимі операції. За замовчуванням Visual Basic виконує перевірку при кожному обчисленні із змінною типу даних з плаваючою комою (Single і Double), аби гарантувати, що результуюче значення знаходиться всередині необхідного інтервалу. Якщо значення має неправильну величину, то видається помилка. Виконується також контроль на недопустимі операції. Вибір даної опції відключає цей контроль, що може прискорити обчислення з плаваючою комою. Однак в цьому випадку можуть мати місце неправильні результати без індикації помилок.
Remove Safe Pentium FDIV Checks	Відключає генерацію спеціального кода, який підвищує безпеку ділення з плаваючою комою (FDIV – floating-point division) на процесорах Pentium, що мають помилку при цій операції. Компілятор внутрішнього кода автоматично додає додатковий код для виконання операцій ділення з плаваючою комою на процесорах Pentium, які мають помилку FDIV. Вибір цієї опції дозволяє зменшити та прискорити код, але в інколи може створювати на таких процесорах невеликі погрешності.
Allow Unrounded Floating-point Operations	Дозволяє компілятору порівнювати результати виразів з плаваючою комою без попереднього округлення цих результатів до правильної точності. При обчисленнях з плаваючою комою перед виконанням порівняння значення як правило округлюються до відповідного ступеня точності (Single або Double). Вибір цієї опції дозволяє компілятору, коли можна виконати операцію більш ефективно, проводити порівняння з плаваючою комою до заокруглення. Це підвищує швидкість деяких операцій з плаваючою комою. Однак це може призвести до того, що обчислення будуть проводитись з більш високою точністю, чим очікується, і два значення з плаваючою комою, які могли б вважатись рівними, не будуть такими.

З компіляцією проблем виникнути не повинно.

Висновки

От ми з Вами і дійшли до завершення нашого невеличкого курсу.

Я спробував розглянути основні принципи програмування на Visual Basic.

Сподіваюсь, що мої уроки допоможуть Вам освоїти ази програмування на Visual Basic.