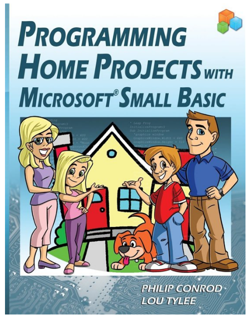
# Programmeren van thuisprojecten met Microsoft Small Basic: Hoofdstuk 6: Flash Card Math Quiz Project

Dit hoofdstuk is een bewerking van het boek Programming Home Projects with Microsoft Small Basic van Philip Conrod en Lou Tylee.

Om dit boek in zijn geheel te kopen, zie de Computer Science For Kids website .


PROGRAMMING HOME PROJECTS WITH MICROSOFT® SMALL BASIC
PHILIP CONROD · LOU TYLEE

## 6. Flash Card Math Quiz Project

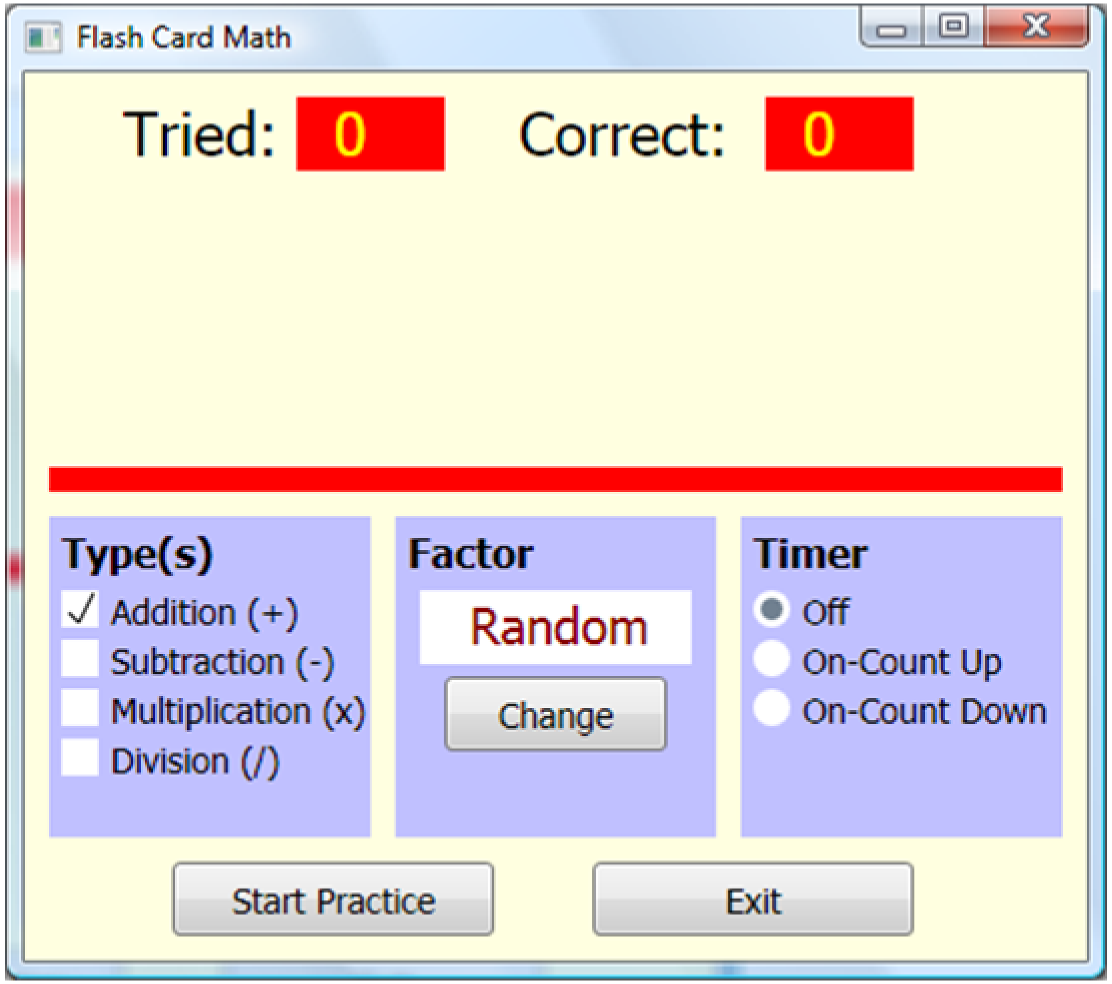
### Bekijken en bekijken

In dit hoofdstuk bouwen we een project waarmee kinderen (of volwassenen) hun basisvaardigheden voor optellen, aftrekken, vermenigvuldigen en delen kunnen oefenen. Met het **Flash Card Math Quiz Project** kunt u het probleemtype selecteren, welke nummers u wilt gebruiken en heeft u drie timingopties. We kijken ook naar het gebruik van willekeurige getallen en het accepteren van toetsenbordinvoer.

### Flash Card Math Quiz Project Preview

In dit hoofdstuk zullen we een **flash card wiskundeprogramma** bouwen. Willekeurige wiskundige problemen (selecteerbaar uit optellen, aftrekken, vermenigvuldigen en/of delen) met behulp van de getallen van 0 tot 9 worden gepresenteerd. Timingopties zijn beschikbaar om zowel nauwkeurigheid als snelheid op te bouwen.

Het voltooide project wordt opgeslagen als **FlashCard** in de map **HomeSB\HomeSB Projects\FlashCard**. Start Small Basic en open het voltooide project. **Voer** het project uit (klik op **Uitvoeren** op de werkbalk of druk op <**F5**>). Het flash card wiskundeprogramma wordt weergegeven als:



Tried: 0 Correct: 0

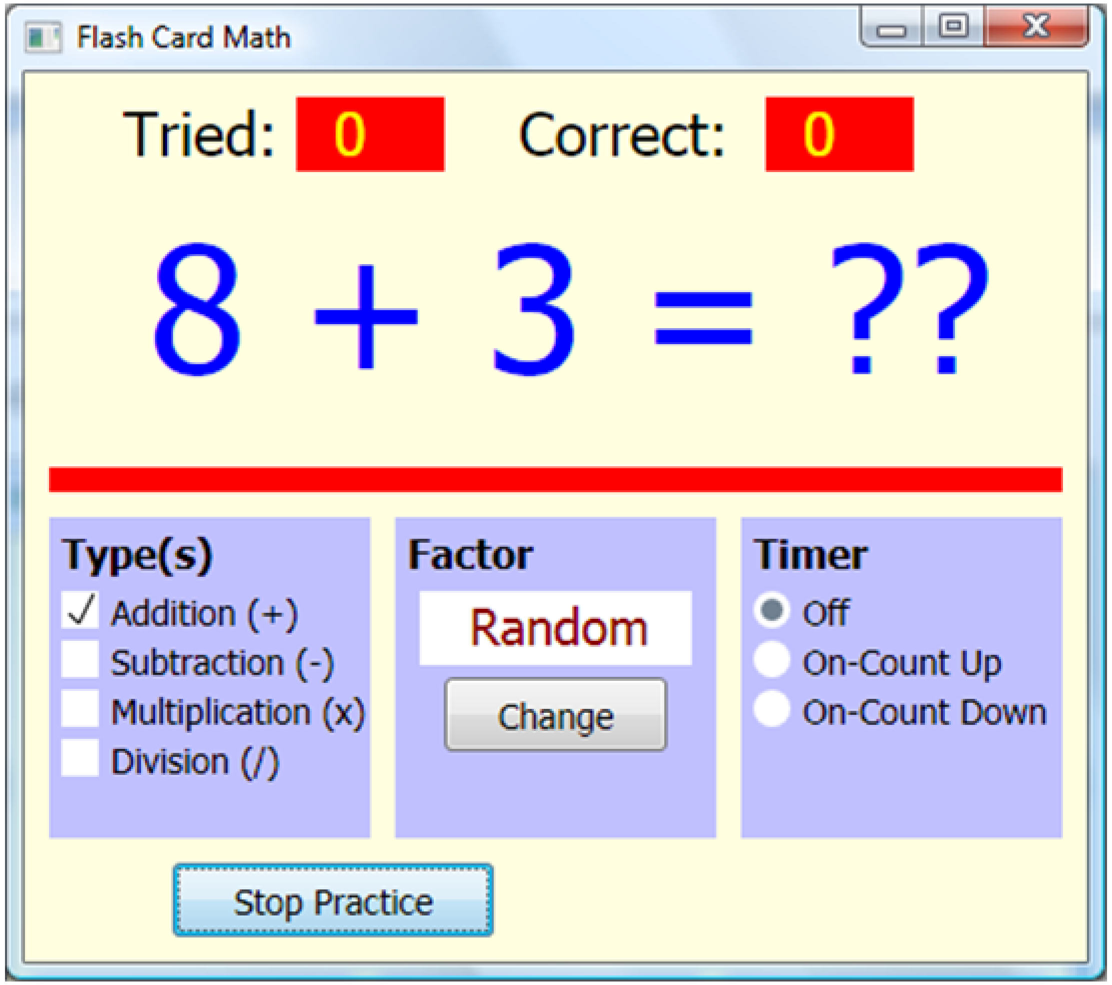
✓ Addition (+)

Random

● Off

Er zijn veel opties beschikbaar. Kies eerst probleemtype in het vak **Type**. Kies uit problemen **met optellen**, **aftrekken**, **vermenigvuldigen** en/of **delen** (klik op uw keuze; u kunt meer dan één probleemtype kiezen). Kies uw **factor** (gebruik de knop **Wijzigen**), een willekeurig getal van 0 tot 9 of kies **Willekeurig** voor willekeurige factoren. Deze opties kunnen op elk moment worden gewijzigd. Om wiskundige feiten te oefenen, klikt u op de knop **Oefenen starten**.

Wanneer ik op **Praktijk starten** klik (met de standaardkeuzes), zie ik:


Flash Card Math

Tried: 0 Correct: 0

8 + 3 = ??

Type(s): [✓ Addition (+)] Subtraction (-) Multiplication (x) Division (/)

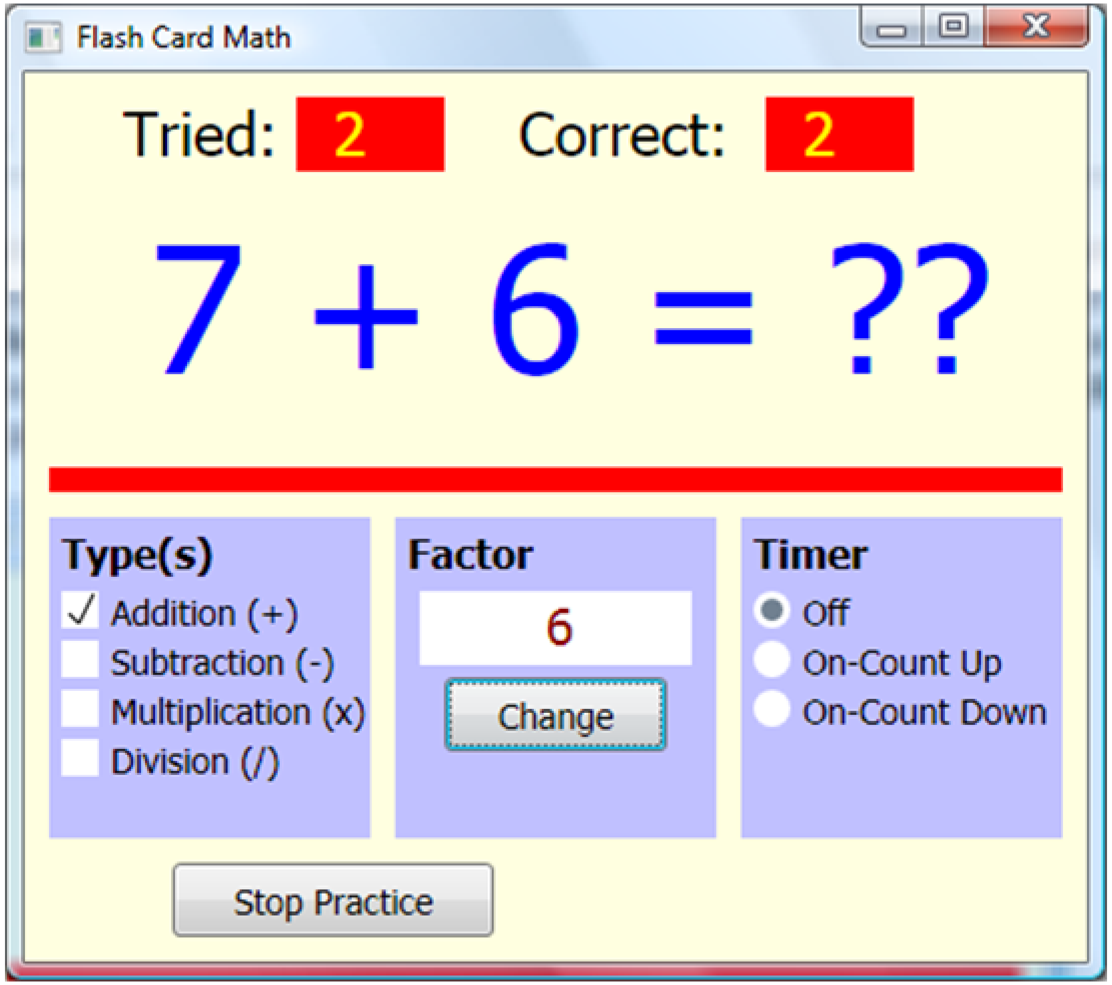
Factor: Random

Timer: [✓ Off] On-Count Up On-Count Down

Je kunt nu zien dat er een groot display in het midden is waar het probleem (8 + 3 =) wordt weergegeven. Het programma wacht op een antwoord op dit probleem. Typ uw antwoord. Als het correct is, wordt het getal naast **Correct:** verhoogd. Of het nu klopt of niet, er wordt een ander probleem gepresenteerd.

Een paar opmerkingen over het invoeren van uw antwoord. Het primaire doel van het programma is om snelheid op te bouwen bij het oplossen van eenvoudige problemen. Als zodanig heb je één kans om een antwoord in te voeren - er is geen wissen. Als het antwoord meer dan twee cijfers heeft (het aantal cijfers in het antwoord wordt weergegeven met vraagtekens), typt u uw antwoord van links naar rechts. Als het antwoord bijvoorbeeld 10 is, typt u een 1 en vervolgens een 0. Probeer verschillende toevoegingsproblemen om te zien hoe antwoorden worden ingevoerd. U kunt op elk gewenst moment stoppen met het oefenen van wiskundige problemen door op de knop **Oefenen stoppen** te klikken.

Andere probleemtypen kunnen op elk moment worden geselecteerd en een nieuwe factor worden gekozen. Elk probleem wordt willekeurig gegenereerd, op basis van probleemtype en factorwaarde. Voor **Addition** krijgt u problemen met het gebruik van uw factor als tweede addend. Als u **6** als uw factor kiest (klik op **Wijzigen** totdat **6** verschijnt als de **factor**), is een voorbeeldprobleem:



Flash Card Math

Tried: 2 Correct: 2

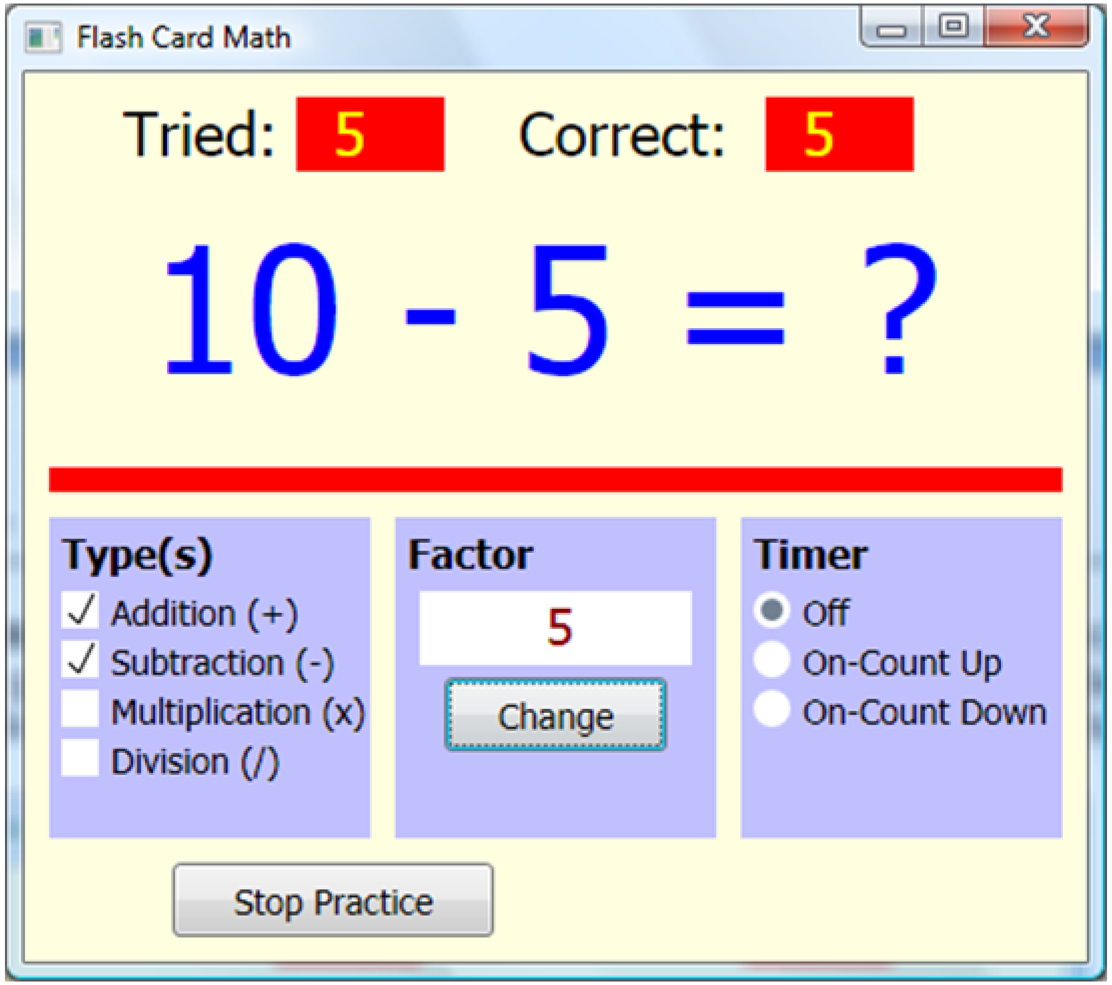
7 + 6 = ??

Type(s)
- [✓ Addition (+)]
- Subtraction (-)
- Multiplication (x)
- Division (/)

Factor: 6

Timer
- [✓ Off]
- On-Count Up
- On-Count Down

Voor **Aftrekken** (klik op **Aftrekken** onder **Type** - let op het toegevoegde vinkje) krijgt u problemen met het gebruik van uw factor als de subtrahend (het getal dat wordt afgetrokken). Als u een factor **5 selecteert**, is een voorbeeld van een aftrekprobleem:


Flash Card Math

Tried: 5 Correct: 5

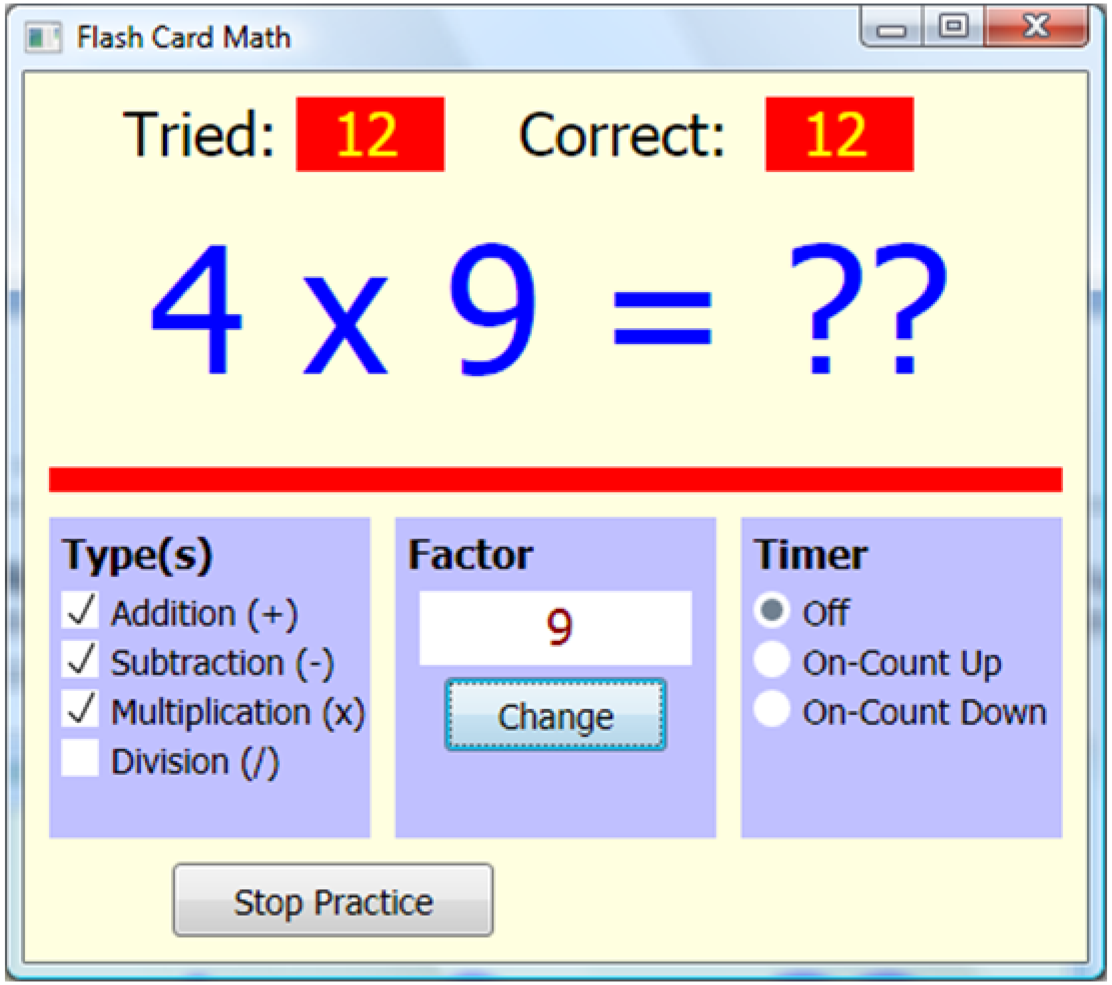
10 - 5 = ?

Type(s)
- [✓] Addition (+)
- [✓] Subtraction (-)
- [ ] Multiplication (x)
- [ ] Division (/)

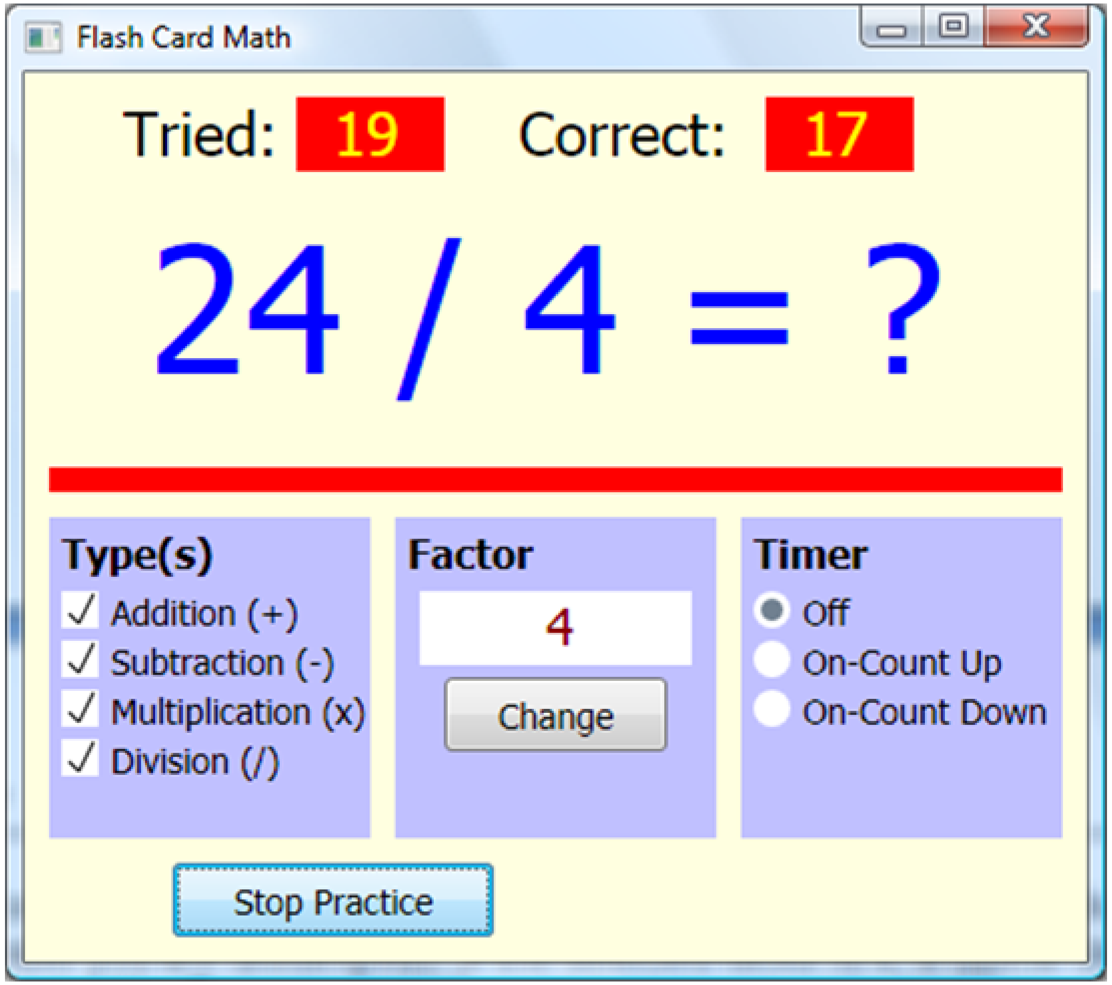
Factor: 5

Timer
- ● Off
- ○ On-Count Up
- ○ On-Count Down

Voor **Vermenigvuldiging** krijg je problemen met het gebruik van je factor als vermenigvuldiger (het getal waarmee je vermenigvuldigt). Als een factor **9** is geselecteerd, is een voorbeeld van een vermenigvuldigingsprobleem:



Ten slotte krijg je voor **Divisie** problemen met het gebruik van je factor als deler (het getal waarmee je deelt).
Als de geselecteerde factor 4 is, is een typisch verdelingsprobleem:


Flash Card Math

Tried: 19 Correct: 17

24 / 4 = ?

**Type(s)**
- ✓ Addition (+)
- ✓ Subtraction (-)
- ✓ Multiplication (x)
- ✓ Division (/)

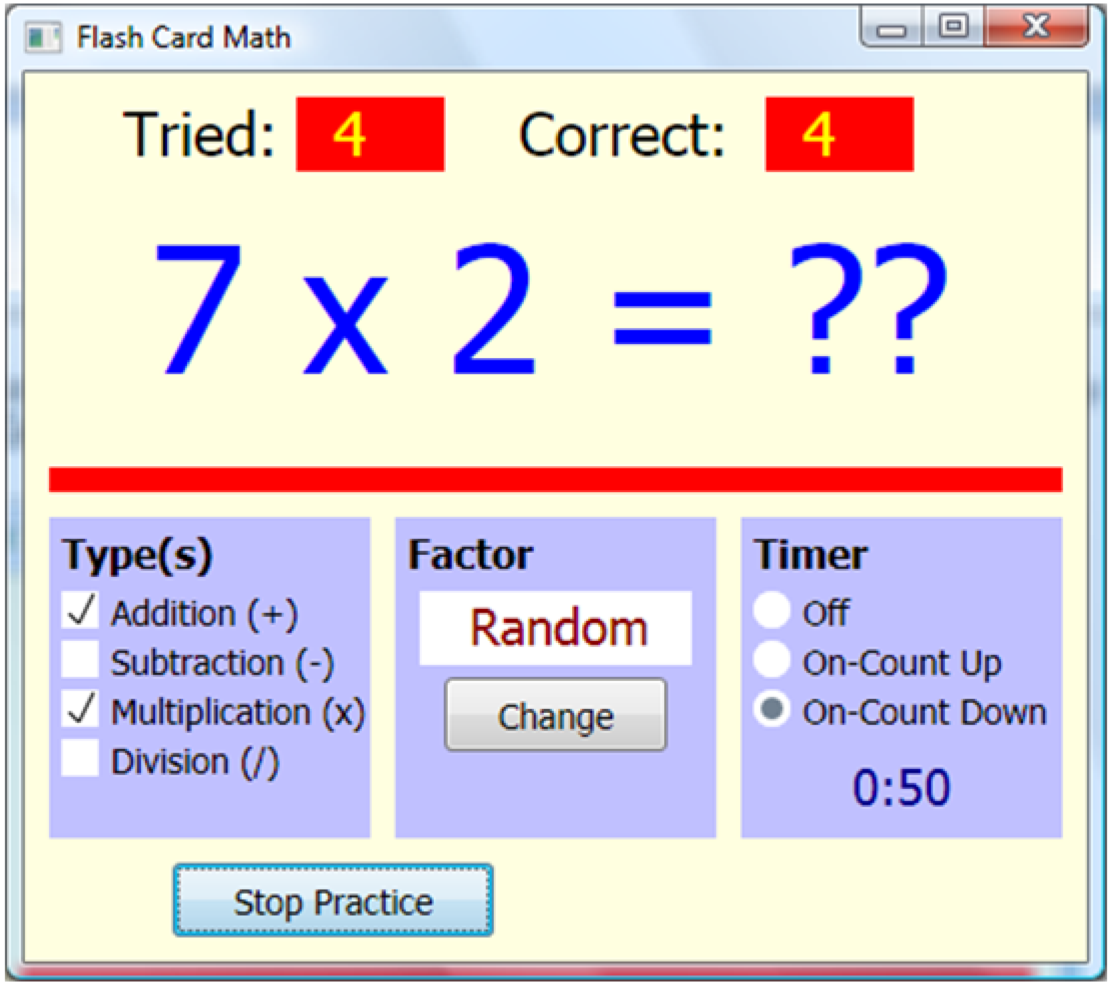
**Factor**
4

**Timer**
- [✓ Off]
- On-Count Up
- On-Count Down

Zoals gezegd hoef je geen specifieke factor te kiezen – **Er** kunnen willekeurige factoren gekozen worden. Probeer allerlei factoren met allerlei soorten problemen.

Er is nog een andere optie om te overwegen bij het gebruik van het wiskundige project van de flash-kaart - de overeenkomstige optiekeuzes bevinden zich in het vak **Timer**. Deze opties kunnen alleen worden geselecteerd als er geen problemen worden opgelost. Er zijn hier drie keuzes. Als u **Uit** selecteert, lost u problemen op totdat u op **Oefenen stoppen** klikt. Als u **On-Count Up** selecteert, verschijnt er een timer en houdt de computer bij hoe lang u problemen hebt opgelost (maximaal 30 minuten is toegestaan). Als u **On-Count Down** selecteert, verschijnt er een timer, samen met +/- knoppen. De knoppen worden gebruikt om in te stellen hoe lang je problemen wilt oplossen (maximaal 30 minuten is toegestaan). De timer telt dan af, zodat u problemen kunt oplossen totdat de toegewezen tijd is verstreken.

Probeer desgewenst de timeropties. Hier is het begin van een run die ik heb gemaakt met de **optie On-Count Down** (vanaf 1 minuut):


Flash Card Math

Tried: 4 Correct: 4

7 x 2 = ??

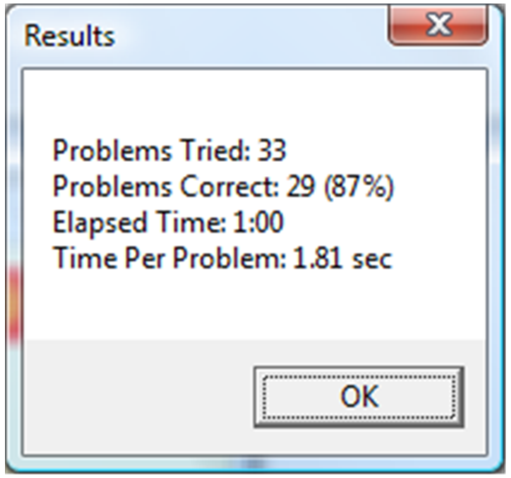
Type(s)
- [✓ Addition (+)]
- Subtraction (-)
- [✓ Multiplication (x)]
- Division (/)

Factor
Random

Timer
- Off
- On-Count Up
- [✓ On-Count Down]

0:50

Zodra je klaar bent met het oefenen van wiskundige problemen (je klikte op **Stop oefenen** of de tijd was op met de optie **Aftellen**), verschijnt er een berichtvenster met de resultaten van je kleine quiz. Dit vak vertelt je hoeveel problemen je hebt opgelost en hoeveel je correct hebt gekregen (inclusief een percentagescore). Als de timer aanstond, krijg je ook te horen hoe lang je problemen aan het oplossen was en hoeveel tijd (gemiddeld) je aan elk probleem besteedde. Hier is het berichtenvak dat ik zag toen ik klaar was met de quiz die ik hierboven begon:

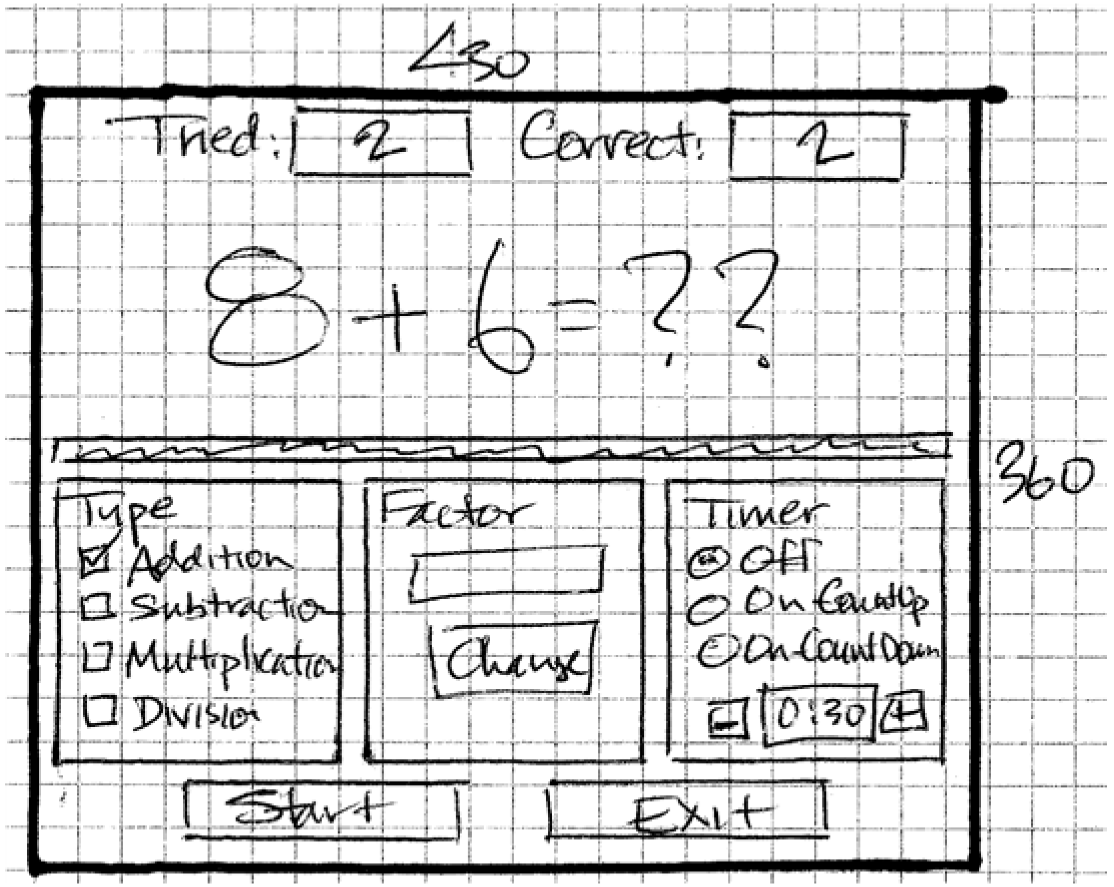

```
Problems Tried: 33
Problems Correct: 29 (87%)
Elapsed Time: 1:00
Time Per Problem: 1.81 sec
```

Klik op **OK** en u kunt het opnieuw proberen. Klik op de knop **Afsluiten** in **Flash Card Math** wanneer u klaar bent met het oplossen van problemen.

Je bouwt dit project nu in verschillende fasen. We richten ons eerst op het **ontwerp van vensters**. We bespreken de besturingselementen die worden gebruikt om het venster te bouwen en de initiële eigenschappen vast te stellen. En we behandelen **codeontwerp** in detail. We behandelen willekeurige generatie problemen, selectie van de verschillende programma-opties en hoe timing te gebruiken.

## Flash Card Math Window Ontwerp

Hier is de schets voor de vensterlay-out van het **Flash Card Math-programma**:



Er is hier veel. Twee gelabelde tekstvormen worden gebruikt voor het scoren. Er is een grote tekstvorm in het midden van het venster die wordt gebruikt om het wiskundige probleem weer te geven. Een verdelende rechthoek scheidt de probleemweergave van optiekeuzes en knoppen. Twee knopbesturingselementen (onderaan het venster) worden gebruikt om de problemen te starten en te stoppen en om het project te verlaten. Er zijn drie rechthoeken die worden gebruikt om optiekeuzes te groeperen. De eerste bevat vier "selectievakjes" die worden gebruikt om het probleemtype te selecteren. De tweede bevat een knop en display die worden gebruikt om nummers te selecteren die in de problemen worden gebruikt. Het derde vak bevat drie kleine cirkels die worden gebruikt om de timingoptie te selecteren. Twee kleine knopbedieningen (gemarkeerd met – en +) worden gebruikt om de hoeveelheid tijd aan te passen die wordt gebruikt in de flashkaartboren.

We zullen beginnen met het schrijven van code voor de applicatie. We schrijven de code in verschillende stappen. We beginnen met de code die nodig is om alle bovenstaande elementen aan het programmavenster toe te voegen. Het programma wordt gebouwd in de standaardconfiguratie: **toevoegingsproblemen**, **willekeurige** factor en timer **uit**. Later voegen we code toe om deze standaardwaarden te wijzigen.

Start een nieuw programma in Small Basic. Eenmaal gestart, raden we u aan het programma onmiddellijk op te slaan met een naam die u kiest. Hiermee stelt u de map- en bestandsstructuur in die nodig is voor uw programma.

## Vensterontwerp - Scoren en probleemweergave

We beginnen met het instellen van het grafische venster en het toevoegen van de tekstvormen die het aantal geprobeerde problemen weergeven (**TriedDisplay**) en het correct beantwoorde aantal (**CorrectDisplay**). We voegen ook een lege tekstvorm toe (**ProblemDisplay**, met een zeer groot lettertype) om het probleem te beantwoorden en tekenen een rode, verdelende rechthoek. Voeg deze code toe aan uw programma:

```
' Flash Card Wiskunde

InitializeProgram()

Sub InitializeProgram

  'grafisch venster

GraphicsWindow. Breedte = 430

GraphicsWindow. Hoogte = 360

GraphicsWindow. Titel = "Flash Card Math"

GraphicsWindow. BackgroundColor = "LightYellow"

'labels/scores'

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. FontBold = "onwaar"

GraphicsWindow. Fontsize = 24

GraphicsWindow. DrawText(40, 10, "Geprobeerd:")

GraphicsWindow. DrawText(200, 10, "Correct:")

GraphicsWindow. BrushColor = "Rood"
```

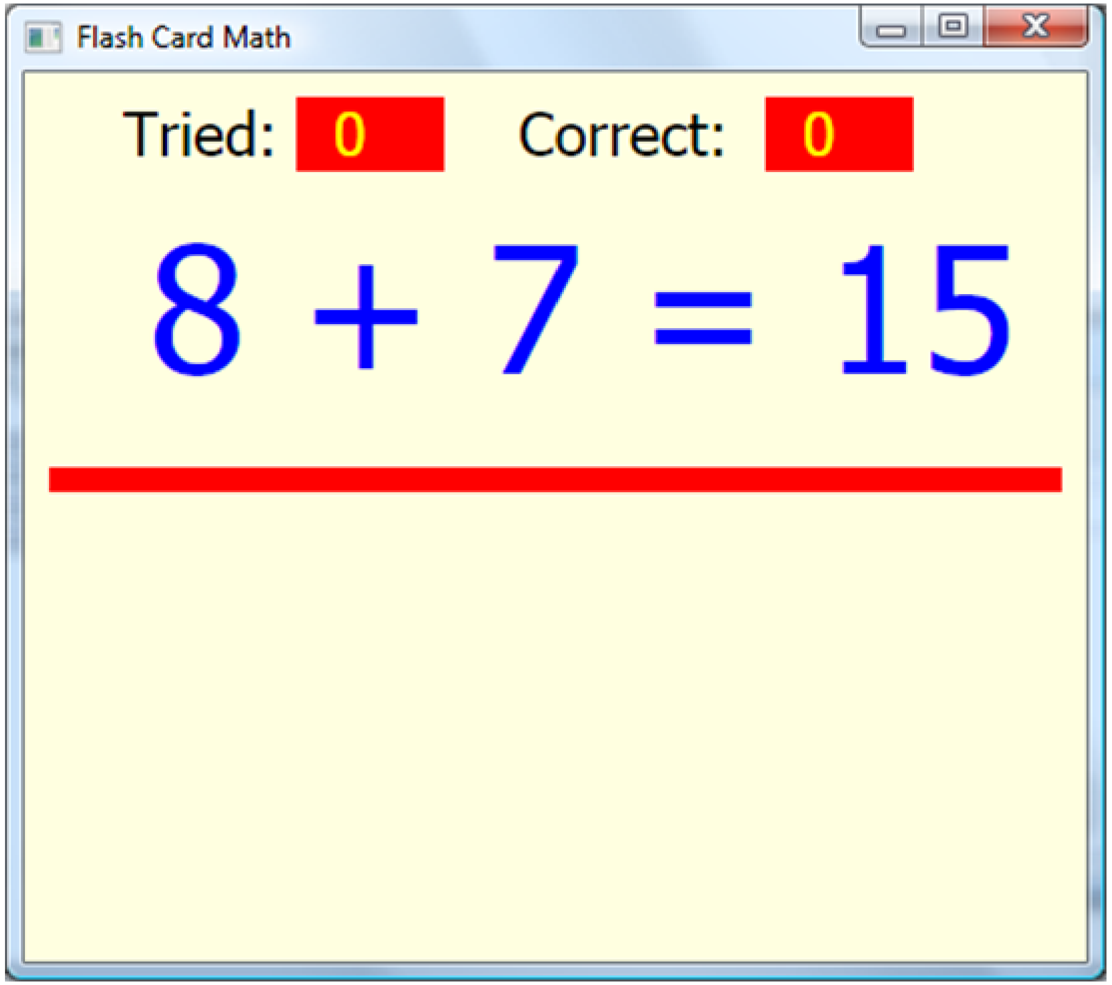
```
    GraphicsWindow. FillRectangle(110, 10, 60, 30)

    GraphicsWindow. FillRectangle(300, 10, 60, 30)

    GraphicsWindow. BrushColor = "Geel"

    TriedDisplay = Vormen. AddText("0")

    Vormen. Move (TriedDisplay, 125, 10)

    CorrectDisplay = Vormen. AddText("0")

    Vormen. Verplaatsen (CorrectDisplay, 315, 10)

    'probleemweergave

    GraphicsWindow. BrushColor = "Blauw"

    GraphicsWindow. Fontsize = 72

    ProblemDisplay = Vormen. AddText("")

    Vormen. Verplaatsen (ProblemDisplay, 50, 50)

    'scheidingswand

    GraphicsWindow. BrushColor = "Rood"

    GraphicsWindow. FillRectangle(10, 160, 410, 10)

    Eindsub
```

De eerste regel code roept een subroutine **InitializeProgram** aan waar we alle code plaatsen die nodig is om het programma in te stellen voor gebruik. Alle resterende code hier gaat in die subroutine.

Zoals geschreven, zult u geen probleem zien omdat **ProblemDisplay** leeg is. Als u een idee wilt krijgen van hoe een probleem eruit zal zien, wijzigt u de regelinstelling van de tekst tijdelijk in:

```
    ProblemDisplay = Vormen. Tekst toevoegen("8 + 7 = 15")
```

Sla **nu op** en **voer** het programma uit. U zou moeten zien:



Merk op hoe we de tekstvormen (**TriedDisplay** en **CorrectDisplay**) op rode rechthoeken plaatsen om ze wat achtergrond te geven. Je vraagt je misschien af hoe we **FontSize** van **72** wisten te gebruiken in de probleemweergave. Er was geen magie - we probeerden gewoon verschillende waarden totdat we een mooie weergave kregen.

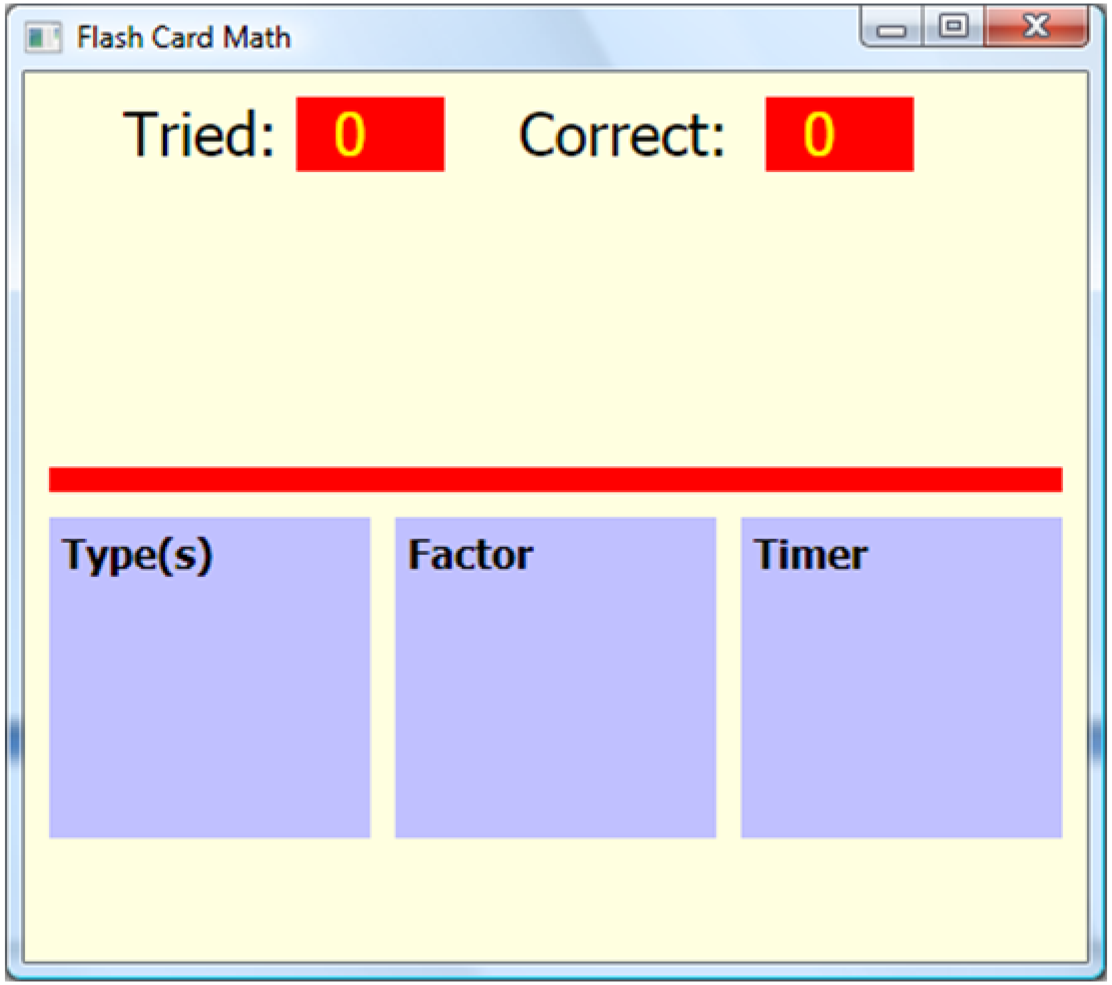
Vergeet niet om de regelinstelling van de probleemweergave terug te zetten naar:

```
    ProblemDisplay = Vormen. AddText("")
```

Vervolgens beginnen we met het schrijven van de code om de programma-opties te kiezen. Voordat u dit doet, voegt u deze code toe (in **InitializeProgram**) om de drie blauwe rechthoeken te tekenen en te labelen om elke optiekeuze vast te houden:

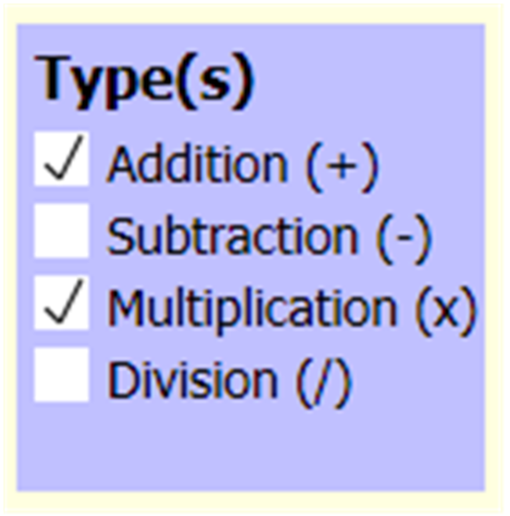
```
'probleemtypes/factor/timer

GraphicsWindow. BrushColor = GraphicsWindow. GetColorFromRGB(192, 192, 255)

GraphicsWindow. FillRectangle(10, 180, 130, 130)

GraphicsWindow. FillRectangle(150, 180, 130, 130)

GraphicsWindow. FillRectangle(290, 180, 130, 130)

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. Fontsize = 16

GraphicsWindow. FontBold = "waar"

GraphicsWindow. DrawText(15, 185, "Type(s)")

GraphicsWindow. DrawText(155, 185, "Factor")

GraphicsWindow. DrawText(295, 185, "Timer"))
```

**Opslaan** en **uitvoeren** om de toegevoegde rechthoeken te zien:



## Vensterontwerp - Probleemtypen kiezen

In het **Flash Card Math-programma** kunt u een van de vier elementaire rekenkundige bewerkingen oefenen: optellen, aftrekken, vermenigvuldigen en / of delen. We willen de gebruiker een handige manier bieden om te kiezen uit de probleemtypen. Hier is een weergave van de gekozen methode:



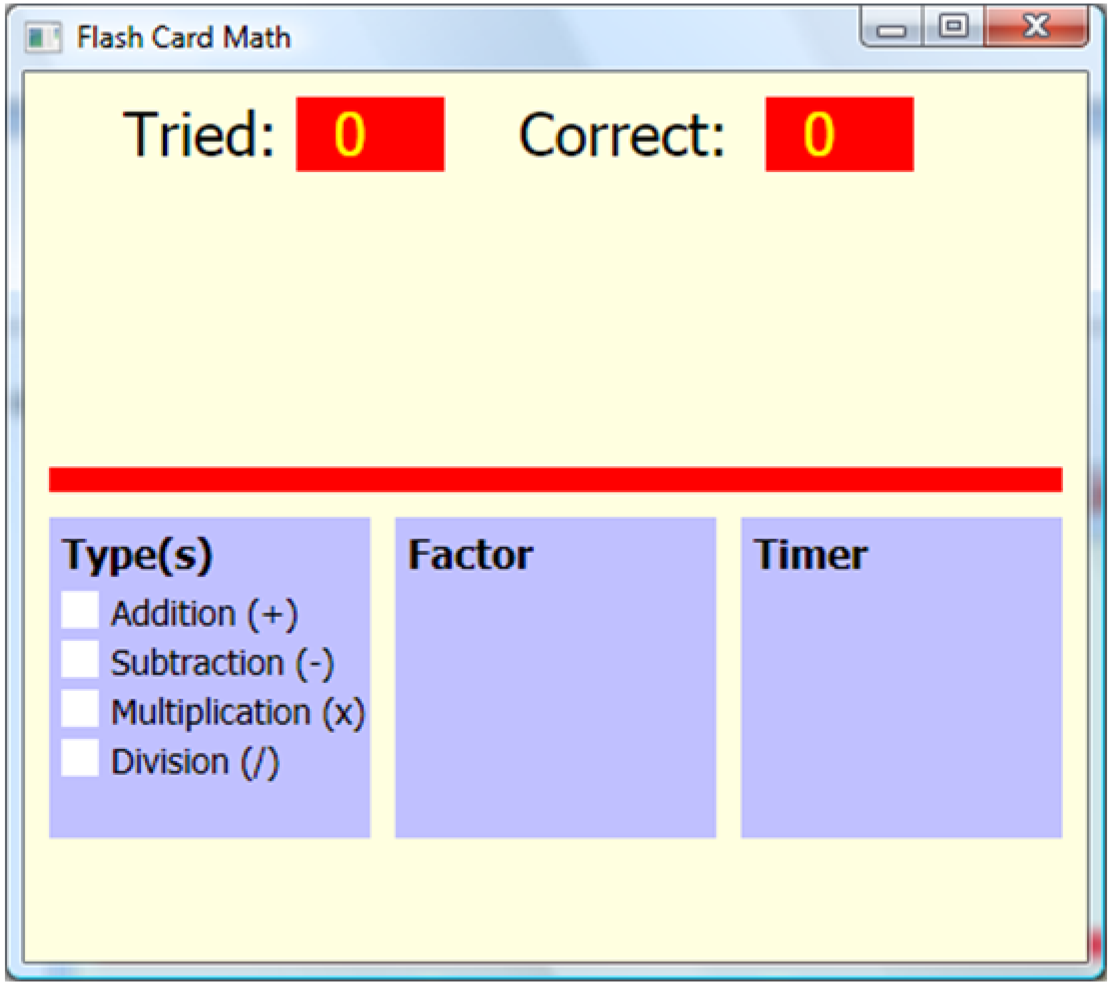
Naast de keuzes worden witte vierkantjes getekend. Wanneer een gebruiker op een vierkantje klikt en daar geen vinkje staat, verschijnt er een (waarbij dat probleemtype wordt geselecteerd). Als er al een vinkje staat, verdwijnt het (deselecteren van dat probleemtype).

De code om het selectieproces te implementeren is een beetje ingewikkeld en zal later worden besproken. Voorlopig tekenen we het scherm ervan uitgaande dat alleen **toevoegingsproblemen** (standaardkeuze) beschikbaar zijn. Voeg deze code toe aan **InitializeProgram**:

```
'soorten problemen'

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. Fontsize = 14

GraphicsWindow. FontBold = "onwaar"

GraphicsWindow. DrawText(35, 210, "Toevoeging (+)")

GraphicsWindow. DrawText(35, 230, "Aftrekken (-)")

GraphicsWindow. DrawText(35, 250, "Vermenigvuldiging (x)")

GraphicsWindow. DrawText(35, 270, "Divisie (/)")

GraphicsWindow. BrushColor = "Wit"

GraphicsWindow. FillRectangle(15, 210, 15, 15)

GraphicsWindow. FillRectangle(15, 230, 15, 15)
GraphicsWindow. FillRectangle(15, 250, 15, 15)

GraphicsWindow. FillRectangle(15, 270, 15, 15)

ProblemSelected[1] = "waar"

ProblemSelected[2] = "false"

ProblemSelected[3] = "false"

ProblemSelected[4] = "false"
```

Deze code tekent vier witte vierkanten met de juiste labels. Een array **ProblemSelected** wordt gebruikt om op te geven welke probleemtypen worden geselecteerd. Voorlopig is alleen element 1 (**Optelproblemen**) "**waar**".

**Sla** het programma op en **voer** het uit. De selectievierkanten worden weergegeven:



Er verschijnt geen vinkje in het vierkant dat aangeeft dat **Toevoegingsproblemen** zijn geselecteerd. We zullen dat corrigeren wanneer we code schrijven voor het selectieproces.

## Raamontwerp - Factorselectie

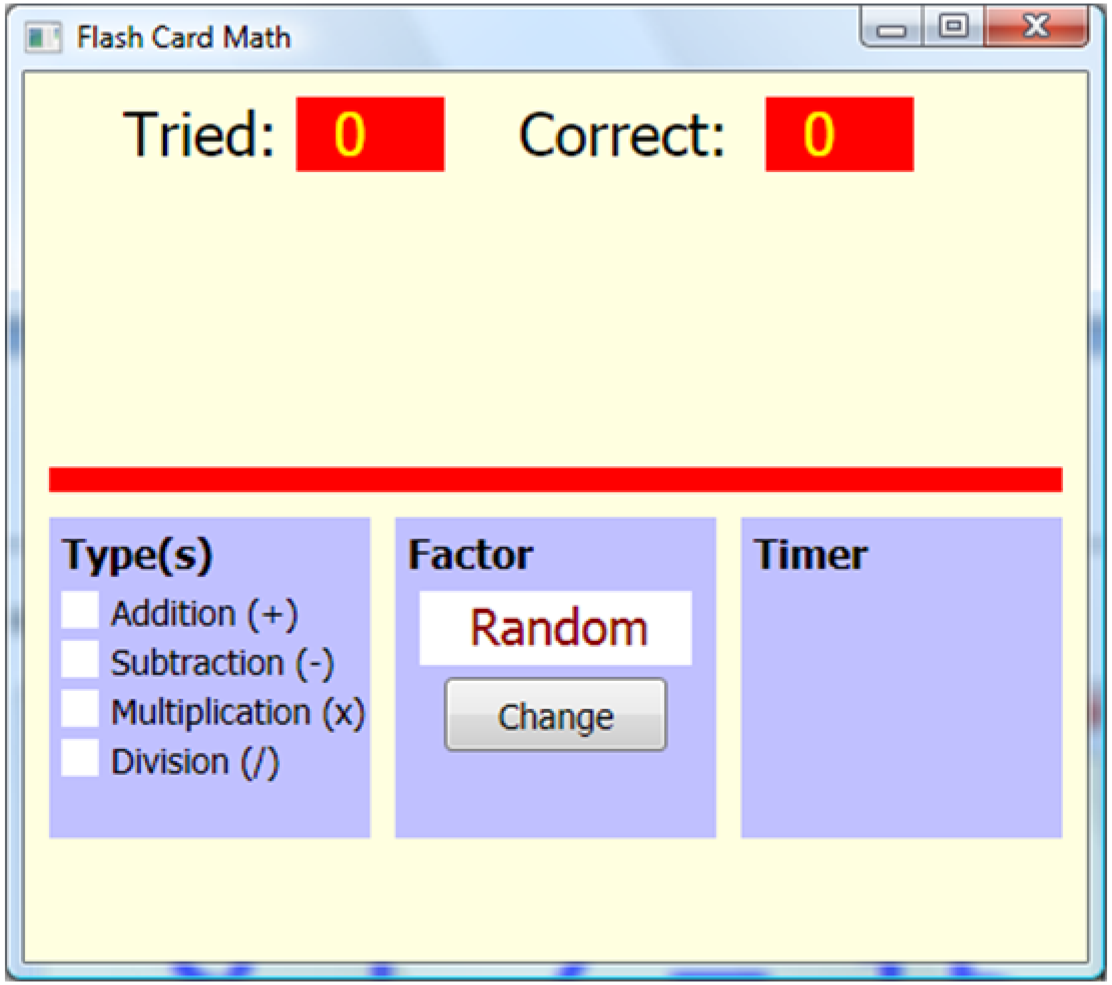
Een gebruiker kan een bepaald getal (factor) van 0 tot 9 kiezen om wiskundige problemen te oefenen of kan een willekeurige factor kiezen. Deze keuze wordt gemaakt in het **keuzevak Factor**. Net als bij de selectie van het probleemtype, stellen we eenvoudig het venster in voor de standaardkeuze (**willekeurige** factor) en richten we ons later op de code voor het wijzigen van de factor.

Voeg deze code toe aan **InitializeProgram**:

```
"factor

GraphicsWindow. BrushColor = "Wit"

GraphicsWindow. FillRectangle(160, 210, 110, 30)

GraphicsWindow. BrushColor = "DarkRed"

GraphicsWindow. Fontsize = 20

FactorDisplay = Vormen. AddText("Willekeurig")

Vormen. Verplaatsen (FactorDisplay, 180, 212)

FactorChoice = -1

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. Fontsize = 14

FactorButton = Besturingselementen. AddButton("Verandering", 170, 245)

Bediening. Setgrootte (FactorButton, 90, 30)
```

Hiermee wordt een tekstvorm (**FactorDisplay**) boven op een witte rechthoek geplaatst om aan te geven welke factor is geselecteerd (toont **voorlopig Willekeurig**). Een knopbediening (**FactorButton**) wordt gebruikt om dit display te wijzigen. De variabele **FactorChoice** houdt de geselecteerde factor bij. Als het positief is, is de waarde de geselecteerde factor (0 tot 9). Wanneer de waarde -1 is (de standaardwaarde hier), wordt een willekeurige factor gebruikt.

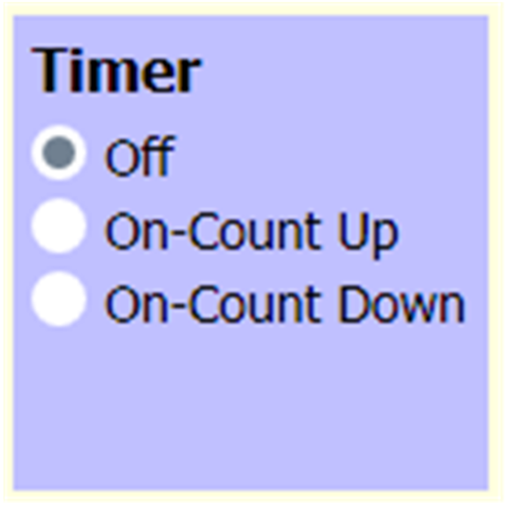
**Sla het programma op** en **voer** het uit om het factorselectievak te zien:



Later schrijven we code om de weergegeven factor te wijzigen wanneer op de knop **Wijzigen** wordt geklikt.

## Vensterontwerp - Timeropties

In het **Flash Card Math-programma** kunt u uw oefening timen als u dat wilt. Er zijn drie keuzes: timer **Uit**, timer **Aan – Count Up**, timer **On – Count Down**. Net als de selectie van het probleemtype, willen we de gebruiker een handige manier bieden om de timeroptie te kiezen. Het verschil tussen deze selectie en die van probleemtype is dat er maar één keuze gemaakt kan worden. Hier is een weergave van de gekozen methode:



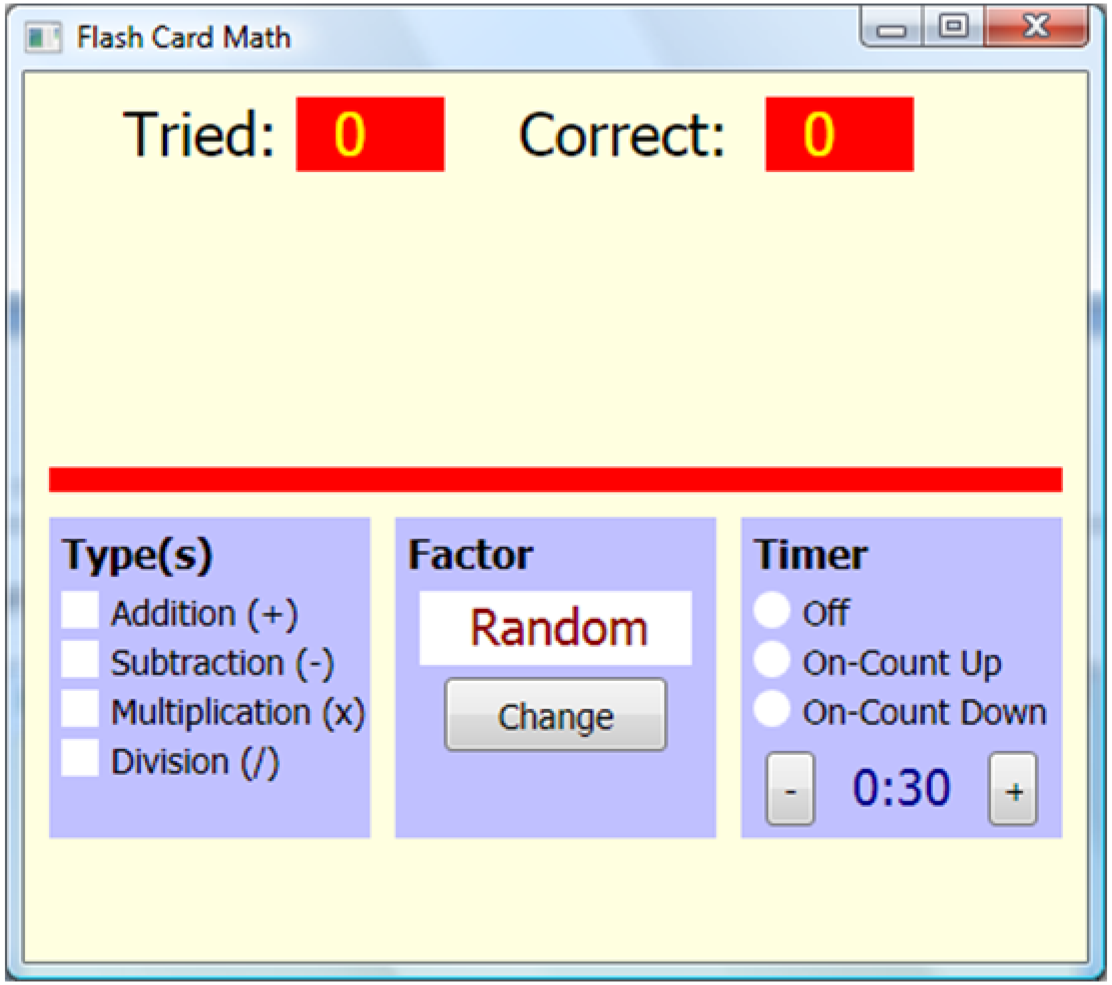
Naast de keuzes worden witte cirkels getekend. Wanneer een gebruiker op een cirkel klikt, wordt die cirkel gemarkeerd (door die timeroptie te selecteren). Alle andere keuzes worden 'ongemarkeerd'. Wanneer een van de timer **Aan-opties** is geselecteerd, worden timinginformatie en knoppen om tijden te wijzigen weergegeven onder de cirkels.

Net als bij de selectie van het probleemtype, is ook de code achter het selecteren van timingopties betrokken en zal deze later worden besproken. Hier schrijven we code om het scherm te bouwen en gebruiken we de standaardoptie timer **uit**. Voeg deze code toe aan **InitializeProgram**:

```
'timer keuzes

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. Fontsize = 14

GraphicsWindow. FontBold = "onwaar"

GraphicsWindow. DrawText(315, 210, "Uit")

GraphicsWindow. DrawText(315, 230, "On-Count Up")

GraphicsWindow. DrawText(315, 250, "Aftellen")

GraphicsWindow. BrushColor = "Wit"

GraphicsWindow. FillEllips(295, 210, 15, 15)

GraphicsWindow. FillEllips(295, 230, 15, 15)
GraphicsWindow. FillEllips(295, 250, 15, 15)

TimerChoice = 0 ' 0-uit, 1 -aan/omhoog, 2- aan/omlaag

GraphicsWindow. BrushColor = "DarkBlue"

GraphicsWindow. Fontsize = 20

TimeDisplay = Vormen. AddText ("0:30")

Vormen. Verplaatsen (TimeDisplay, 335, 277)

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. Fontsize = 14

TimerPlusButton = Bediening. AddButton("+", 390, 275)

Bediening. SetSize (TimerPlusButton, 20, 30)

TimerMinusButton = Besturingselementen. AddButton("-", 300, 275)

Bediening. SetSize (TimerMinusButton, 20, 30)
```

Deze code tekent de cirkels (weglatingstekens) naast de gelabelde keuzes. Een variabele **TimerChoice** slaat op welke optie is geselecteerd (0-uit, 1-op, aftellen, 2-op, aftellen). Het is in eerste instantie ingesteld op nul (timer is uit). Een tekstvorm (**TimeDisplay**) wordt gebruikt om de tijd weer te geven wanneer de timer wordt gebruikt en twee knoppen (**TimerPlusButton**, **TimerMinusButton**) worden gebruikt om de timer aan te passen in de aftelsituatie.

**Sla** het programma op en **voer** het uit om de nieuwe toevoegingen te zien:



Er wordt geen selectiemarkering weergegeven in de cirkel die aangeeft dat de **timer is uitgeschakeld**. We zullen dat corrigeren wanneer we code schrijven voor het selectieproces. **TimeDisplay** en de twee aanpassingsknoppen mogen niet worden weergegeven in de standaardconfiguratie (timer **uit**), dus voeg deze drie regels toe in **InitializeProgram** (nadat de code zojuist is toegevoegd om ze te maken) om deze besturingselementen te verbergen:

```
Vormen. HideShape (TimeDisplay)

Bediening. HideControl (TimerMinusButton)

Bediening. HideControl (TimerPlusButton)
```

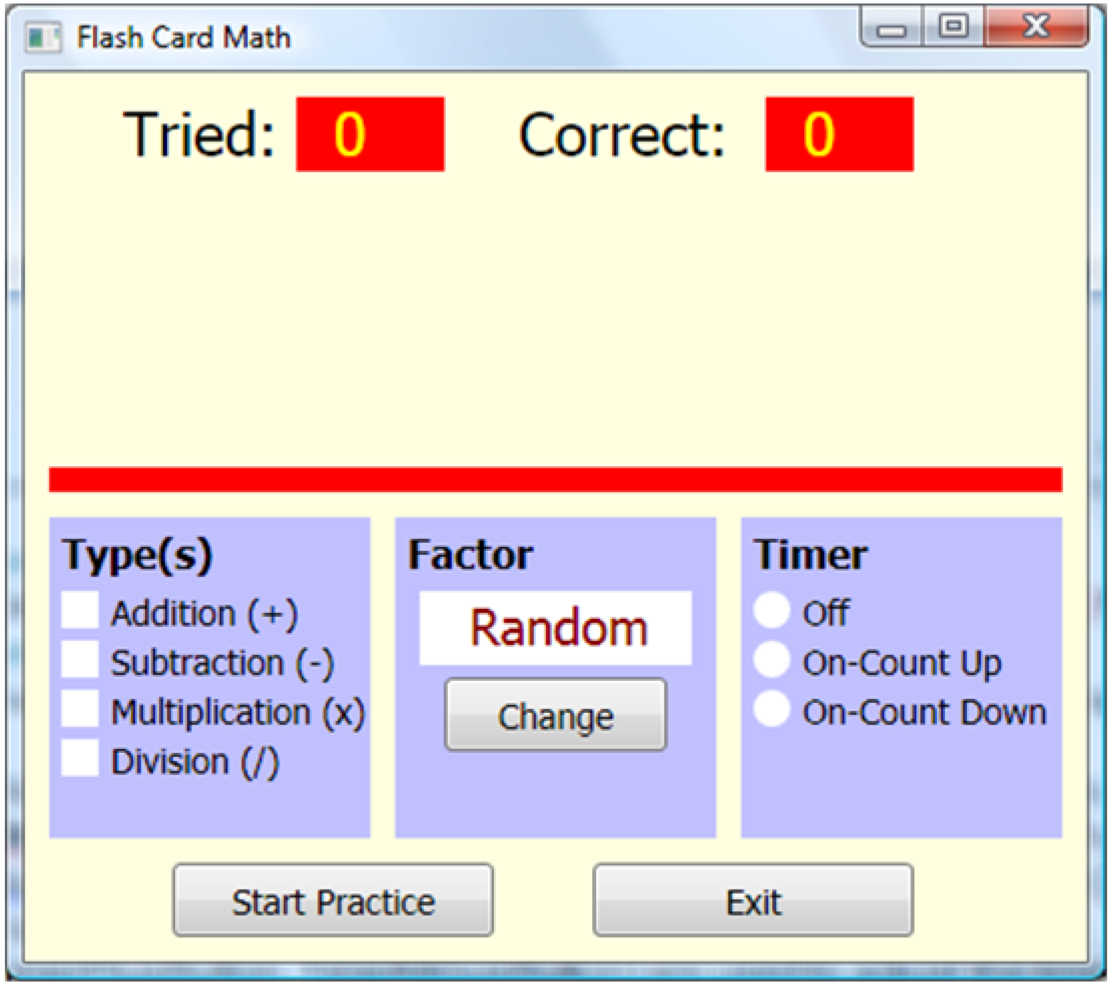
Voer het programma opnieuw uit om ervoor te zorgen dat de besturingselementen verborgen zijn.

## Vensterontwerp - Knoppen toevoegen

Het laatste element dat nodig is in het venster zijn de twee knoppen om de werking van het programma te regelen. Eén knop (**StartStopButton**) wordt gebruikt om het oefenen van wiskundige problemen te starten en te stoppen. Eén knop (**ExitButton**) wordt gebruikt om het programma te stoppen. Voeg deze code toe aan **InitializeProgram**:

```
"knoppen

GraphicsWindow. BrushColor = "Zwart"

GraphicsWindow. Fontsize = 14

StartStopButton = Controls.AddButton("Start Practice", 60, 320)

Controls.SetSize(StartStopButton, 130, 30)

ExitButton = Controls.AddButton("Exit", 230, 320)

Controls.SetSize(ExitButton, 130, 30)
```

**Save** and **Run** the program to see the final window design:



We now start writing the code behind choosing all the options and generating and solving problems. As a first step, we write the code that generates a random problem and gets the answer from the user, updating the score.

## Code Design – Start Practice

The idea of the flash card math project is to display a problem, receive an answer from the user and check for correctness. Problems can be of four different types with different factor choices and different timer options. For now, we work with **Addition** problems with **Random** factors. And, we will ignore the timer options. Once this initial code is working satisfactorily, other problem types and factors and timing will be considered. Again, this step-by-step approach to building a project is far simpler than trying to build everything at once.

Things begin by clicking the **Start Practice** button (**StartStopButton**). When this happens, the following steps are taken:

- Change caption of **StartStopButton** to **Stop Practice**.
- Hide **ExitButton**.
- Set number of problems tried (**NumberTried**) and number correct (**NumberCorrect**) to zero.
- Generate and display a problem in **ProblemDisplay**.
- Obtain answer from user.
- Check answer and update score.

Once each generated problem is answered, subsequent problems are generated and answered.

The user answers problems until he/she clicks **Stop Practice** (or time elapses in timed drills). The steps followed at this point are:

- Change caption of **StartStopButton** to **Start Practice**
- Show **ExitButton**.
- Clear **ProblemDisplay**.
- Present results.

The code behind the listed steps is fairly straightforward. First, add this line at the end of **InitializeProgram** to allow detection of button clicks:

```
Controls.ButtonClicked = ButtonClickedSub
```

And the corresponding subroutine (**ButtonClickedSub**) called when a button is clicked:

```
Sub ButtonClickedSub

  B = Controls.LastClickedButton

  If (B = StartStopButton) Then

    StartStopButtonClicked()

  ElseIf (B = ExitButton) Then

    Program.End()

  EndIf
```

```
EndSub
```

We have added possibilities for clicking on either **StartStopButton** or **ExitButton**. As seen, if a user clicks the **ExitButton**, the program simply ends.

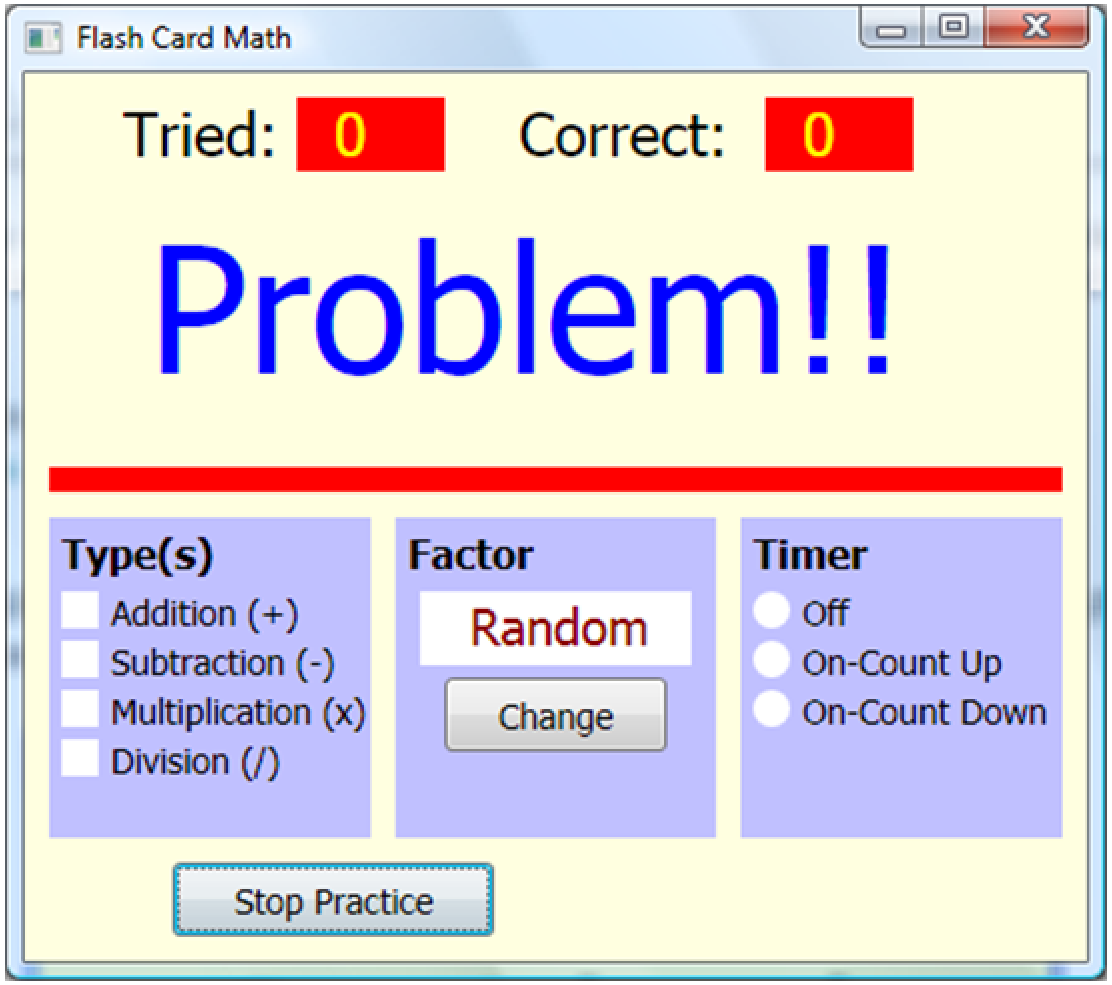
Now, use this code in the **StartStopButtonClicked** subroutine (implements the steps above, except for presenting results):

```
Sub StartStopButtonClicked

  If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then

    Controls.SetButtonCaption(StartStopButton, "Stop Practice")

    Controls.HideControl(ExitButton)

    NumberTried = 0

    NumberCorrect = 0

    Shapes.SetText(TriedDisplay, "0")

    Shapes.SetText(CorrectDisplay, "0")

    GetProblem()

  Else

    Controls.SetButtonCaption(StartStopButton, "Start Practice")

    Controls.ShowControl(ExitButton)

    Shapes.SetText(ProblemDisplay, "")

  EndIf

EndSub
```

This code uses a subroutine **GetProblem** to generate the random problem and display it in **ProblemDisplay**. Add this nearly empty procedure (we'll fill it in soon).

```
Sub GetProblem

  Shapes.SetText(ProblemDisplay, "Problem!!")

EndSub
```

**Save** and **Run** the project. Click **Start Practice** to make sure buttons change as planned. You will also see the generated "problem":



Now, click **Stop Practice**. Make sure **Exit** works.

This framework seems acceptable. We continue code design by discussing **problem generation** and **obtaining an answer** (including **scoring**) from the user. Then, later we discuss **timing** and **presenting the results**.

## Code Design – Problem Generation

To generate a problem, we examine the current options selected by the user and produce a random problem based on these selections. All code will be in the **GetProblem** subroutine currently in the framework code. And, even though we are only using **Addition** problems with **Random** factors in this initial design, we will write code for all possibilities.

The steps involved in generating a random flash card problem are:

- Select problem type (random selection based on checked choices in **Type** box)
- Generate factor (based on selection in **Factor** box)
- Formulate problem and determine correct answer.
- Display problem in **ProblemDisplay** text shape, replacing correct answer with question marks (?) in place of digits. An example of the desired form of the returned value is:

  8 + 6 = ??

  where question marks tell the user how many digits are in the correct answer.

- Initialize **YourAnswer** to blank and **DigitNumber** (used to check if you have typed in all the digits to the answer).

Let's look at each step of the problem generation process. The first step is to choose a random problem type from the maximum of four possibilities. We will use a simple approach, first generating a random number from 1 to 4 (1 representing addition, 2 representing subtraction, 3 representing multiplication and 4 representing division). If the corresponding element of the **ProblemSelected** array is "**true**" (meaning that check box is checked), that will be the problem type. It that element of **ProblemSelected** is "**false**", we choose another random number. We continue this process until a problem type is selected. Notice this approach assumes at least one check box is always selected. We will make sure this is the case when developing code for the problem type option. There are more efficient ways to choose problem type which don't involve loops, but, for this simple problem, this works quite well.

A snippet of code that performs the choice of problem type (**ProblemType**) is:

```
ProblemType = 0

While (ProblemType = 0)

  P = Math.GetRandomNumber(4)

  If (P = 1 And ProblemSelected[1]) Then

    'Addition

    ProblemType = P
```

```
      ElseIf (P = 2 And ProblemSelected[2]) Then
        'Subtraction
        ProblemType = P
      ElseIf (P = 3 And ProblemSelected[3]) Then
        'Multiplication
        ProblemType = P
      ElseIf (P = 4 And ProblemSelected[4]) Then
        'Division
        ProblemType = P
      EndIf
    EndWhile
```

Once a problem type is selected, we determine the factor used to generate a problem. It can be a selected value from 0 to 9, or a random value from 0 to 9, based on the value selected in the **Factor** box. For now, we assume that value is provided by a subroutine **GetFactor** that determines a **Factor** value, based on problem type **ProblemType**.

Each problem has four variables associated with it: **Factor**, representing the value returned by **GetFactor**, **Number**, the other number used in the math problem, **CorrectAnswer**, the problem answer, and **Problem**, a string representation of the unsolved problem. Once a problem type and factor have been determined, we find values for each of these variables. Each problem type has unique considerations for problem generation. Let's look at each type.

For **Addition** problems (**ProblemType = 1**), the selected factor is the second **addend** in the problem. The string form of addition problems (**Problem**) will be:

Number + Factor =

where **Number** is a random value from 0 to 9, while recall **Factor** is the selected factor. A snippet of code to generate an addition problem and determine the **CorrectAnswer** is:

```
    Number = Math.GetRandomNumber(10) - 1
    GetFactor()
```

```
CorrectAnswer = Number + Factor

Problem = Number + " + " + Factor + " = "
```

For **Subtraction** problems (**ProblemType = 2**), the factor is the **subtrahend** (the number being subtracted). The string form of subtraction problems (**Problem**) will be:

Number - Factor =

We want all the possible answers to be positive numbers between 0 and 9. Because of this, we formulate the problem in a backwards sense, generating a random answer (**CorrectAnswer**), then computing **Number** based on that answer and the known factor (**Factor**). The code that does this is:

```
GetFactor()

CorrectAnswer = Math.GetRandomNumber(10) - 1

Number = CorrectAnswer + Factor

Problem = Number + " - " + Factor + " = "
```

For **Multiplication** problems (**ProblemType = 3**), the selected factor is the **multiplier** (the number you're multiplying by) in the problem. The string form of multiplication problems (**Problem**) will be:

Number x Factor =

where **Number** is a random value from 0 to 9, and **Factor** is the factor. A snippet of code to generate a multiplication problem and the **CorrectAnswer** is:

```
Number = Math.GetRandomNumber(10) - 1

GetFactor()

CorrectAnswer = Number * Factor

Problem = Number + " x " + Factor + " = "
```

For **Division** problems (**ProblemType = 4**), the factor is the **divisor** (the number doing the dividing). The string form of division problems (**Problem**) will be:

Number / Factor =

Like in subtraction, we want all the possible answers to be positive numbers between 0 and 9. So, we again formulate the problem in a backwards sense, generating a random answer (**CorrectAnswer**), then computing **Number** based on that answer and the known factor (**Factor**). The code that does this is:

```
GetFactor()

CorrectAnswer = Math.GetRandomNumber(10) - 1

Number = CorrectAnswer * Factor

Problem = Number + " / " + Factor + " = "
```

Note with division, we must make sure the factor is never zero (can't divide by zero).

The **GetFactor** routine provides the factor based on the selected value (**FactorChoice**) in the **Factor** box and problem type **ProblemType**. For random factors, it will make sure a zero is not returned if a division problem is being generated. The **GetFactor** subroutine is thus:

```
Sub GetFactor

  If (FactorChoice = -1) Then

    If (ProblemType = 4) Then

      Factor = Math.GetRandomNumber(9)

    Else

      Factor = Math.GetRandomNumber(10) - 1

    EndIf

  Else

    Factor = FactorChoice

  EndIf

EndSub
```

If the Random (**FactorChoice = -1**) option is selected, 0 to 9 is returned for addition, subtraction and multiplication problems; 1 to 9 is returned for division problems (**ProblemType = 4**). In other cases, the selected factor is returned (we will have to make sure zero is not a choice when doing division).

The **GetProblem** subroutine is nearly complete. We want to return the **Problem** variable with appended question marks that represent the number of digits (**NumberDigits**) in the correct answer and we need to initialize **YourAnswer** and **DigitNumber**.  The code snippet that does this is:

```
If (CorrectAnswer < 10) Then

  NumberDigits = 1

  Shapes.SetText(ProblemDisplay, Problem + "?")

Else

  NumberDigits = 2

  Shapes.SetText(ProblemDisplay, Problem + "??")

EndIf

YourAnswer = ""

DigitNumber = 1
```

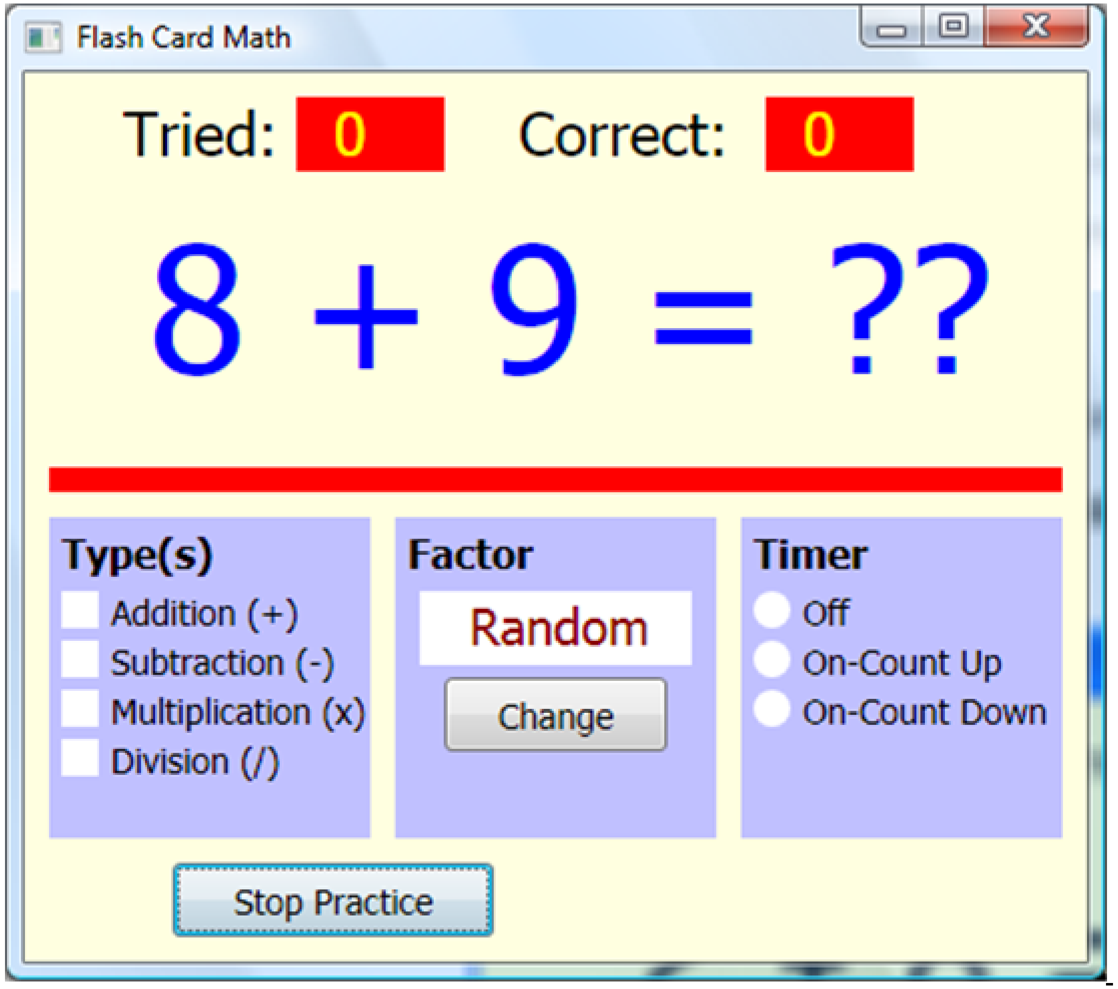
We can now assemble all the little code snippets into a final form for the **GetProblem** subroutine.  To build the **GetProblem** subroutine, first eliminate the single line of code that displays "**Problem!!**".  Then, start with the snippet that selects problem type.  Add each problem generation segment (one for each of the four mathematical operations) in its corresponding location.  Finally, add the question mark appending code.  The finished function is:

```
Sub GetProblem

  ProblemType = 0

  While (ProblemType = 0)

    P = Math.GetRandomNumber(4)

    If (P = 1 And ProblemSelected[1]) Then

      'Addition

      ProblemType = P

      Number = Math.GetRandomNumber(10) - 1

      GetFactor()

      CorrectAnswer = Number + Factor

      Problem = Number + " + " + Factor + " = "
    ElseIf (P = 2 And ProblemSelected[2]) Then
      'Subtraction
      ProblemType = P
      GetFactor()
      CorrectAnswer = Math.GetRandomNumber(10) - 1
      Number = CorrectAnswer + Factor
      Problem = Number + " - " + Factor + " = "
    ElseIf (P = 3 And ProblemSelected[3]) Then
      'Multiplication
      ProblemType = P
      Number = Math.GetRandomNumber(10) - 1
      GetFactor()
      CorrectAnswer = Number * Factor
      Problem = Number + " x " + Factor + " = "
    ElseIf (P = 4 And ProblemSelected[4]) Then
      'Division
      ProblemType = P
      GetFactor()
      CorrectAnswer = Math.GetRandomNumber(10) - 1
      Number = CorrectAnswer * Factor
      Problem = Number + " / " + Factor + " = "
    EndIf
  EndWhile
  If (CorrectAnswer < 10) Then
    NumberDigits = 1
```

```
        Shapes.SetText(ProblemDisplay, Problem + "?")
      Else
        NumberDigits = 2
        Shapes.SetText(ProblemDisplay, Problem + "??")
      EndIf
    EndSub
```

Add this to the project along with the code for **GetFactor**.

**Save** and **Run** the project.  Click **Start Practice** and you should see a random addition problem:



The two question marks tell us there are two digits in the correct answer. We'll see how to get that answer next. At this point, all you can do is click **Stop Practice**. You can then click **Start Practice** to see another addition problem if you'd like. View as many random addition problems as you want.

## Code Design – Obtaining Answer

Once a problem is displayed, the user can enter the digits in the answer. These digits will be entered using the keyboard. The keystrokes will be handled by the graphics window **KeyDown** event.

The steps for obtaining and checking an answer are:

- Make sure keystroke is a number (0 to 9).
- If number, keep keystroke as part of your answer and replace question mark with number.
- If a question mark remains, exit waiting for another keystroke.
- If all question marks are gone, compare your answer (**YourAnswer**) with correct answer (**CorrectAnswer**).
- Increment the number of problems tried.
- If your answer is correct, increment the number of correct problems.
- Update scoring label controls.
- Generate another problem.

Add this line of code at the end of **InitializeProgram** to assign the **KeyDownSub** subroutine to the **KeyDown** event:

```
GraphicsWindow.KeyDown = KeyDownSub
```

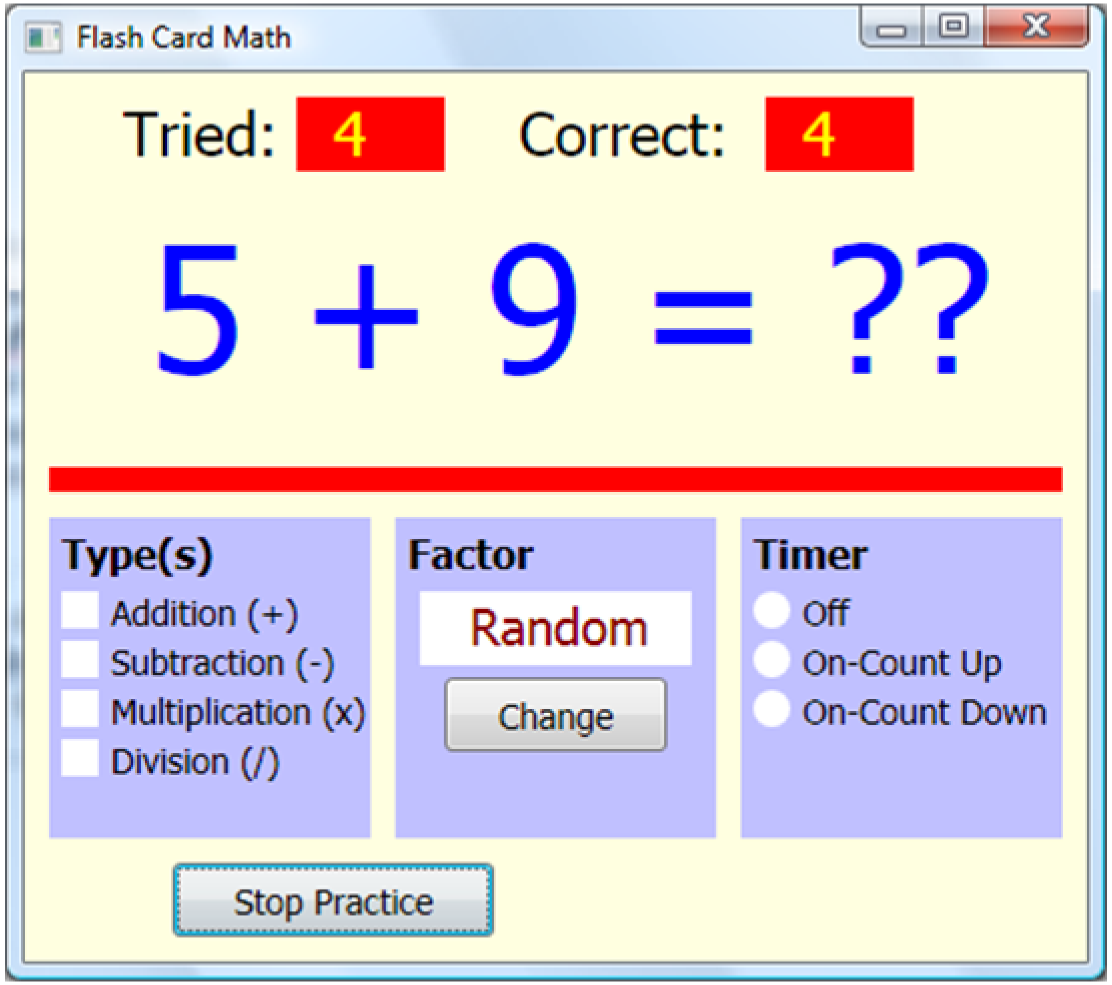
The **KeyDownSub** subroutine that incorporates the steps listed above is then:

```
Sub KeyDownSub

  If (Controls.GetButtonCaption(StartStopButton) = "Stop Practice") Then

    'can only check keystrokes when practicing-only allow number keys

    'number is last character in keypressed

    KeyPressed = Text.GetSubTextToEnd(GraphicsWindow.LastKey, 
Text.GetLength(GraphicsWindow.LastKey))

    If (Text.GetCharacterCode(KeyPressed) >= Text.GetCharacterCode("0") And 
Text.GetCharacterCode(KeyPressed) <= Text.GetCharacterCode("9")) Then

      YourAnswer = Text.Append(YourAnswer, KeyPressed)

      If DigitNumber <> NumberDigits Then

        DigitNumber = DigitNumber + 1

        Shapes.SetText(ProblemDisplay, Problem + YourAnswer + "?")

      Else

        NumberTried = NumberTried + 1

        'check answer
```

```
            If (YourAnswer = CorrectAnswer) Then
              NumberCorrect = NumberCorrect + 1
            EndIf
            Shapes.SetText(TriedDisplay, NumberTried)
            Shapes.SetText(CorrectDisplay, NumberCorrect)
            GetProblem()
          EndIf
        EndIf
      EndIf
    EndSub
```

In the first few lines of code, we make sure we are solving problems before allowing any keystrokes.  Notice how all digits in your answer (represented by the typed character in **KeyPressed**) are saved and concatenated into **YourAnswer**.  Also, notice how the displayed problem is updated, overwriting a question mark, with each keystroke.  As mentioned earlier, the program only gives you one chance to enter an answer - there is no erasing.

**Save** and **Run** the project.  You should now be able to answer as many random addition problems as you'd like.  Try it.  Make sure the score is updating properly.  Here's my window after trying a few problems:



You can stop practicing problems, at any time, by clicking the **Stop Practice** button. Random addition problems will get boring after a while. Let's add the logic for other problem types and other factors.

## Code Design – Choosing Problem Type

The selection of problem type seems simple. Choose the check box or check boxes you want and the correct problem will be generated. But there are a couple of problems we've alluded to. We need to keep a user from "unchecking" all the boxes, leaving no problem type to select. We must make sure at least one box is always selected. And, if **Division** problems are selected, we cannot allow zero (0) to be used as a factor.

Here, we write code to mark and unmark check boxes and to address the first problem (always having one problem selected).  The problem of a zero factor in division problems is addressed when we discuss factors in the next section.  Let's review the location of each check box (all squares with 15 pixel sides).  This will define the 'clickable' areas.

**Addition** – Check box is at (15, 210)
**Subtraction** – Check box is at (15, 230)
**Multiplication** – Check box is at (15, 250)
**Division** – Check box is at (15, 270)

To allow detection of mouse clicks, add this single line of code at the end of **InitializeProgram**:

```
GraphicsWindow.MouseDown = MouseDownSub
```

Now, in the **MouseDownSub** subroutine, we do the following:

- Determine which check box was clicked (**ProblemTypeClicked**).
- If clicked box is checked (**ProblemSelected** is "**true**"), uncheck the box unless the number of boxes checked (**Selections**) is one.  Decrement **Selections** if possible.
- If selected box is unchecked (**ProblemSelected** is "**false**"), check the box and increment **Selections**.

The initial selection is **Addition** problems.  Add this initialization code to **InitializeProgram** after the code setting the initial values of the **ProblemSelected** array.

```
Selections = 1

ProblemTypeClicked = 1

MarkProblemType()
```

Note the initialization code calls a subroutine (**MarkProblemType**) to place an initial check mark next to **Addition**.  That routine is:

```
Sub MarkProblemType

  If (ProblemTypeClicked < 0) Then

    ProblemTypeClicked = Math.Abs(ProblemTypeClicked)

    GraphicsWindow.BrushColor = "White"

    GraphicsWindow.FillRectangle(15, 190 + ProblemTypeClicked * 20, 15, 15)

  Else
```

```
      GraphicsWindow.PenColor = "Black"

      GraphicsWindow.PenWidth = 1

      GraphicsWindow.DrawLine(18, 199 + ProblemTypeClicked * 20, 22, 203 +
  ProblemTypeClicked * 20)

      GraphicsWindow.DrawLine(22, 203 + ProblemTypeClicked * 20, 28, 191 +
  ProblemTypeClicked * 20)

    EndIf

  EndSub
```

This subroutine will place a check mark (drawn using two **DrawLine** methods within the selected check box) when **ProblemTypeClicked** is positive. When **ProblemTypeClicked** is negative, the check mark is removed (the box is cleared).

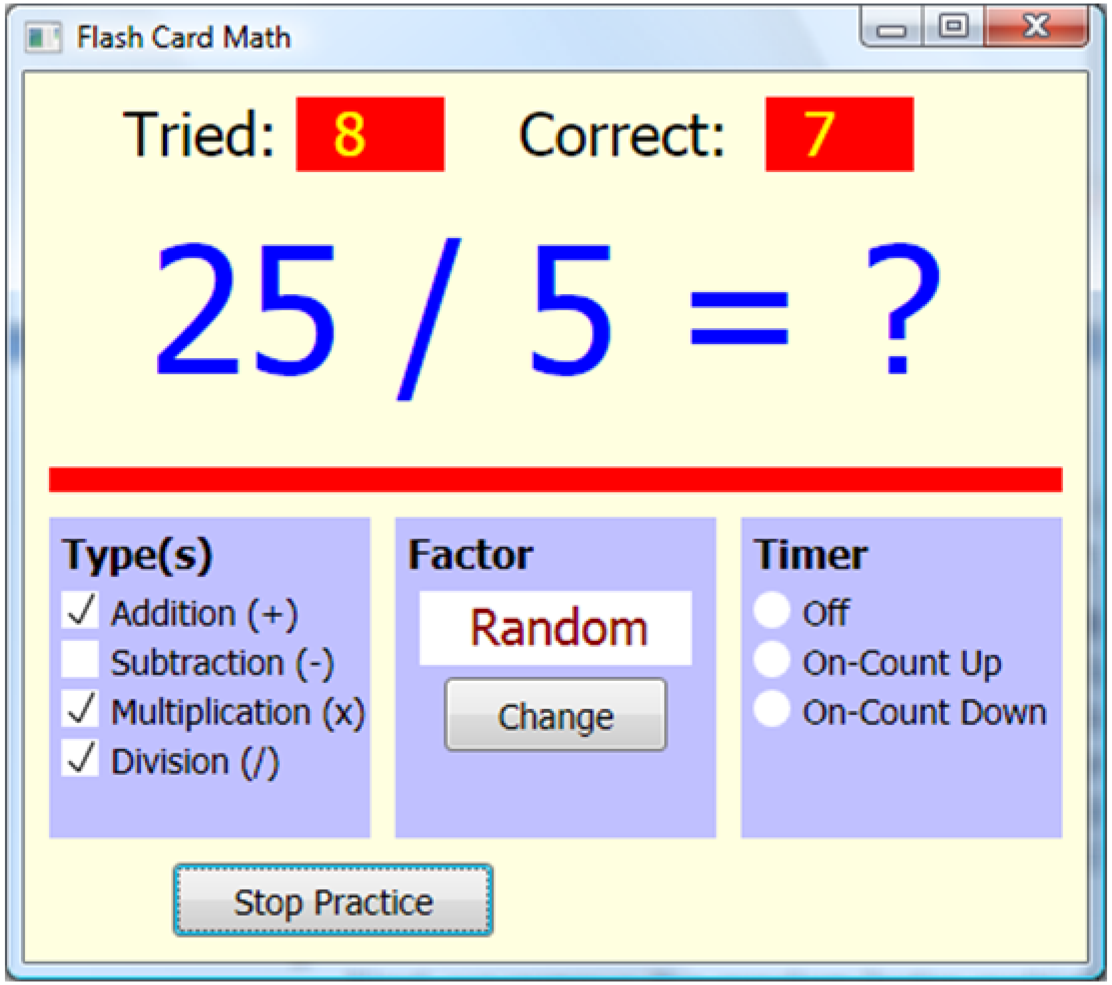
With this information, the **MouseDownSub** procedure that implements the needed steps is:

```
  Sub MouseDownSub

    X = GraphicsWindow.MouseX

    Y = GraphicsWindow.MouseY

    'problem selections - must always have at least one selected

    If (X > 15 And X < 30 And Y > 210 And Y < 285) Then

      If (Y > 210 And Y < 225) Then

       'clicked addition

       ProblemTypeClicked = 1

      ElseIf (Y > 230 And Y < 245) Then

        'clicked subtraction

        ProblemTypeClicked = 2

      ElseIf (Y > 250 And Y < 265) Then

        'clicked multiplication

       ProblemTypeClicked = 3
```

```
        ElseIf (Y > 270 And Y < 285) Then
         'clicked division
          ProblemTypeClicked = 4
        EndIf
        If (ProblemSelected[ProblemTypeClicked] And Selections <> 1) Then
          'clear choice if not last one selected
          Selections = Selections - 1
          ProblemSelected[ProblemTypeClicked] = "false"
          ProblemTypeClicked = -ProblemTypeClicked
          MarkProblemType()
        ElseIf (ProblemSelected[ProblemTypeClicked] = "false") Then
          'mark choice
          Selections = Selections + 1
          ProblemSelected[ProblemTypeClicked] = "true"
          MarkProblemType()
        EndIf
      EndIf
    EndSub
```

You should be able to see how the various steps are implemented. In particular, note when a box is to be cleared, the **ProblemTypeClicked** is converted to a negative number before calling **MarkProblemType**. Enter this code into your project.

**Save** and **Run** the project. Make sure all the newly installed code is doing its job. Try to "uncheck" all the problem type boxes – one box will always remain. You can now solve any problem type. If you change options while solving problems, the changes will be seen once you finish solving the current problem. Try solving problems, changing problem type. Here's my window while solving a division problem (note the check marks):



Next, we remove the random factor restriction.

## Code Design – Changing Factor

Thus far, we have only used a random factor in problems. To add the capability of other factors, we need code for clicking the **FactorButton**. With each click of that button, we cycle through values of **FactorChoice** (0 through 9, -1 for random factors) and display that choice in the text shape (**FactorDisplay**).

Add the shaded code to **ButtonClickSub** to detect clicks of **FactorButton**:

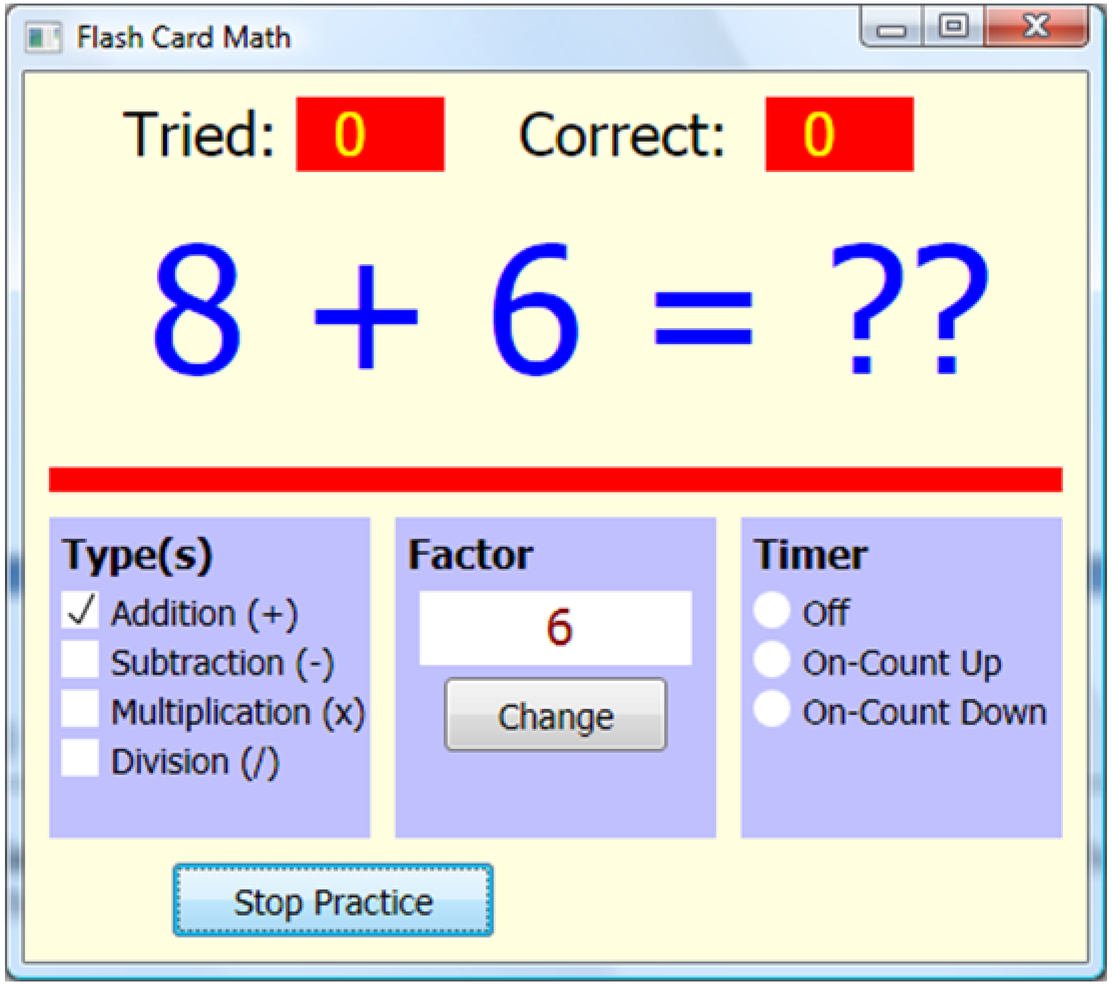
```
Sub ButtonClickedSub
```

```
    B = Controls.LastClickedButton
    If (B = StartStopButton) Then
      StartStopButtonClicked()
    ElseIf (B = ExitButton) Then
      Program.End()
    ElseIf (B = FactorButton) Then
      FactorButtonClicked()
    EndIf
  EndSub
```

Add the **FactorButtonClicked** subroutine to cycle through and display factor values:

```
  Sub FactorButtonClicked
    'change factor choice
    FactorChoice = FactorChoice + 1
    If (FactorChoice > 9) Then
      FactorChoice = -1
      Shapes.SetText(FactorDisplay, "Random")
    Else
      Shapes.SetText(FactorDisplay, Text.Append("     ", FactorChoice))
    EndIf
  EndSub
```

**Save** and **Run** the program. Click **Start Practice**. Click **Change** and watch the values change. Here is my window showing an **Addition** problem with a selected **Factor** of 6:



We need two changes related to not allowing a zero factor for **Division** problems. First, make the shaded change to the **FactorButtonClicked** subroutine:

```
Sub FactorButtonClicked
  'change factor choice
  FactorChoice = FactorChoice + 1
  If (FactorChoice > 9) Then
    FactorChoice = -1
    Shapes.SetText(FactorDisplay, "Random")
```

```
      Else
        If (ProblemSelected[4] And FactorChoice = 0) Then
          'no zero if doing division
          FactorChoice = 1
        EndIf
        Shapes.SetText(FactorDisplay, Text.Append("     ", FactorChoice))
      EndIf
    EndSub
```

Here, if **Division** is a selected problem type and a zero factor is selected, the zero factor will be skipped over.

Second, make the shaded change to the **MouseDownSub** subroutine:

```
    Sub MouseDownSub
      X = GraphicsWindow.MouseX
      Y = GraphicsWindow.MouseY
      'problem selections - must always have at least one selected
      If (X > 15 And X < 30 And Y > 210 And Y < 285) Then
        If (Y > 210 And Y < 225) Then
         'clicked addition
         ProblemTypeClicked = 1
        ElseIf (Y > 230 And Y < 245) Then
          'clicked subtraction
          ProblemTypeClicked = 2
        ElseIf (Y > 250 And Y < 265) Then
          'clicked multiplication
         ProblemTypeClicked = 3
        ElseIf (Y > 270 And Y < 285) Then
```

```
         'clicked division
          ProblemTypeClicked = 4
        EndIf
        If (ProblemSelected[ProblemTypeClicked] And Selections <> 1) Then
          'clear choice if not last one selected
          Selections = Selections - 1
          ProblemSelected[ProblemTypeClicked] = "false"
          ProblemTypeClicked = -ProblemTypeClicked
          MarkProblemType()
        ElseIf (ProblemSelected[ProblemTypeClicked] = "false") Then
          'mark choice
          Selections = Selections + 1
          ProblemSelected[ProblemTypeClicked] = "true"
          MarkProblemType()
          'make sure zero not selected factor if division selected
          If (ProblemTypeClicked = 4 And FactorChoice = 0) Then
            FactorButtonClicked()
          EndIf
        EndIf
      EndIf
    EndSub
```

In this added code, if a user selects **Division** and zero is the factor, we simulate a click on **FactorButton** to increment the factor to one.

**Save** and **Run** the program again. Check **Division** problems. Click on the **Change** button and notice the zero factor never appears. Uncheck **Division** problems. Choose 0 as a factor using the **Change** button. Now, check **Division** again. Notice the factor is changed to 1 and the 0 option does not appear with subsequent clicks on **Change**. You can now solve any problem type with any factor.

## Code Design – Timing Options

Having coded problem generation and answer checking, we can now address the use of timing in the flash card math project. Up to now, we've assumed the timer is off (**TimerChoice = 0**). We have two possibilities for a timer: (1) one where the timer counts up, keeping track how long you are solving problems (**TimerChoice = 1**), and (2) one where the timer counts down from some preset value (**TimerChoice = 2**). In both cases, a text shape (**TimeDisplay**) displays the time in **minutes:seconds** form. In the second case, two button controls (**TimerMinusButton** and **TimerPlusButton**) are used to set the value. The timing will be controlled with a **Timer** object with an interval of 1 second (1000 milliseconds).

We allow changing problem type and factors while solving problems. It wouldn't make sense to be able to change timer options while solving problems – the times would not be correct. We will only allow selection of timer options prior to clicking **Start Practice**. First, we write code to mark the circles next to the three timing options (only one option can be selected). Here are the locations of the circles (all 15 pixel diameters). This will define the 'clickable' areas.

**Off** – Circle is at (295, 210)
**On, Count Up** – Circle is at (295, 230)
**On, Count Down** – Circle is at (295, 250)

Add the shaded code to **ButtonDownSub** to detect the appropriate mouse clicks:

```
Sub MouseDownSub

  X = GraphicsWindow.MouseX

  Y = GraphicsWindow.MouseY

  'problem selections - must always have at least one selected

   .

   .

  If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then

    'timer selections - only one can be selected

    If (X > 295 And X < 310 And Y > 210 And Y < 265) Then

      'problem selections - must always have at least one selected

      If (Y > 210 And Y < 225) Then
```

```
        'clicked timer off
        TimerChoice = 0
        MarkTimerChoice()
      ElseIf (Y > 230 And Y < 245) Then
        'clicked timer on - count up
        TimerChoice = 1
        MarkTimerChoice()
      ElseIf (Y > 250 And Y < 265) Then
        'clicked timer on - count down
        TimerChoice = 2
        MarkTimerChoice()
      EndIf
    EndIf
  EndIf
EndSub
```

Note changes can only be made when **StartStopButton** displays a **Start Practice** caption.

This code uses a subroutine (**MarkTimerChoice**) to mark the selected option and unmark the others. That routine is:

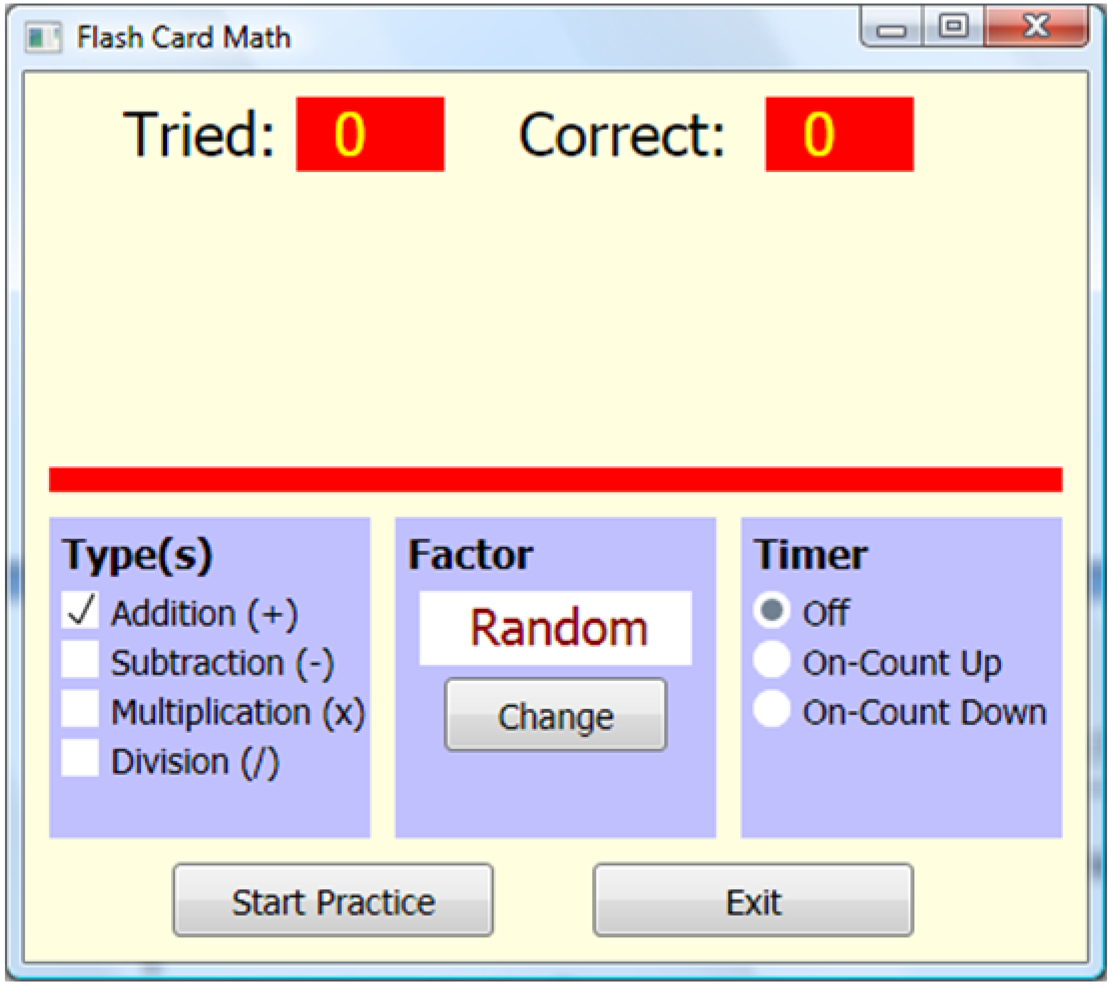
```
Sub MarkTimerChoice
  GraphicsWindow.BrushColor = "White"
  GraphicsWindow.FillEllipse(295, 210, 15, 15)
  GraphicsWindow.FillEllipse(295, 230, 15, 15)
  GraphicsWindow.FillEllipse(295, 250, 15, 15)
  GraphicsWindow.BrushColor = "SlateGray"
```

```
    GraphicsWindow.FillEllipse(298, 213 + TimerChoice * 20, 9, 9)
  EndSub
```

The initial selection is **Off**. Add this line <u>after</u> the line setting the initial value of **TimerChoice** in **InitializeProgram**:

```
  MarkTimerChoice()
```

**Save** and **Run** the program. Notice the initial selection of the **Off** option:



Choose other **Timer** options to make sure proper marking is done.

Other steps must be followed when we switch from one timing option to the next.  We will use a variable (**ProblemTime**) to store the time value (whether counting up or down) in seconds.  When counting down, **ProblemTime** will start at 30 times **TimerIndex** (a value set by the two timer control buttons).  Those steps followed when changing options are:

- If Off (**TimerChoice = 0**) is selected:  hide **TimerDisplay, TimerMinusButton, TimerPlusButton**.
- If On-Count Up (**TimerChoice = 1**) is selected, show **TimerDisplay** and hide **TimerMinusButton** and **TimerPlusButton**.  Initialize **ProblemTime** to 0.  Display **ProblemTime**.
- If On-Count Down (**TimerChoice = 2**) is selected, show **TimerDisplay**, **TimerMinusButton, TimerPlusButton.**  Initialize **ProblemTime** to 30 times **TimerIndex**.  Display **ProblemTime**.

Before attacking this code, add a line to **InitializeProgram** (after the call to **MarkTimerChoice**) setting an initial value for **TimerIndex**:

```
TimerIndex = 1
```

The steps listed above are all implemented in the **MouseDownSub** subroutine.  The needed additions are shaded:

```
Sub MouseDownSub

  X = GraphicsWindow.MouseX

  Y = GraphicsWindow.MouseY

  'problem selections - must always have at least one selected

   .

   .

  If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then

    'timer selections - only one can be selected

    If (X > 295 And X < 310 And Y > 210 And Y < 265) Then

      'problem selections - must always have at least one selected

      If (Y > 210 And Y < 225) Then

        'clicked timer off

        TimerChoice = 0

        MarkTimerChoice()

        Shapes.HideShape(TimeDisplay)
        Controls.HideControl(TimerMinusButton)
        Controls.HideControl(TimerPlusButton)
      ElseIf (Y > 230 And Y < 245) Then
        'clicked timer on - count up
        TimerChoice = 1
        MarkTimerChoice()
        Shapes.ShowShape(TimeDisplay)
        Controls.HideControl(TimerMinusButton)
        Controls.HideControl(TimerPlusButton)
        ProblemTime = 0
        FormatTime()
        Shapes.SetText(TimeDisplay, FormattedTime)
      ElseIf (Y > 250 And Y < 265) Then
        'clicked timer on - count down
        TimerChoice = 2
        MarkTimerChoice()
        Shapes.ShowShape(TimeDisplay)
        Controls.ShowControl(TimerMinusButton)
        Controls.ShowControl(TimerPlusButton)
        ProblemTime = 30 * TimerIndex
        FormatTime()
        Shapes.SetText(TimeDisplay, FormattedTime)
      EndIf
    EndIf
  EndIf
```

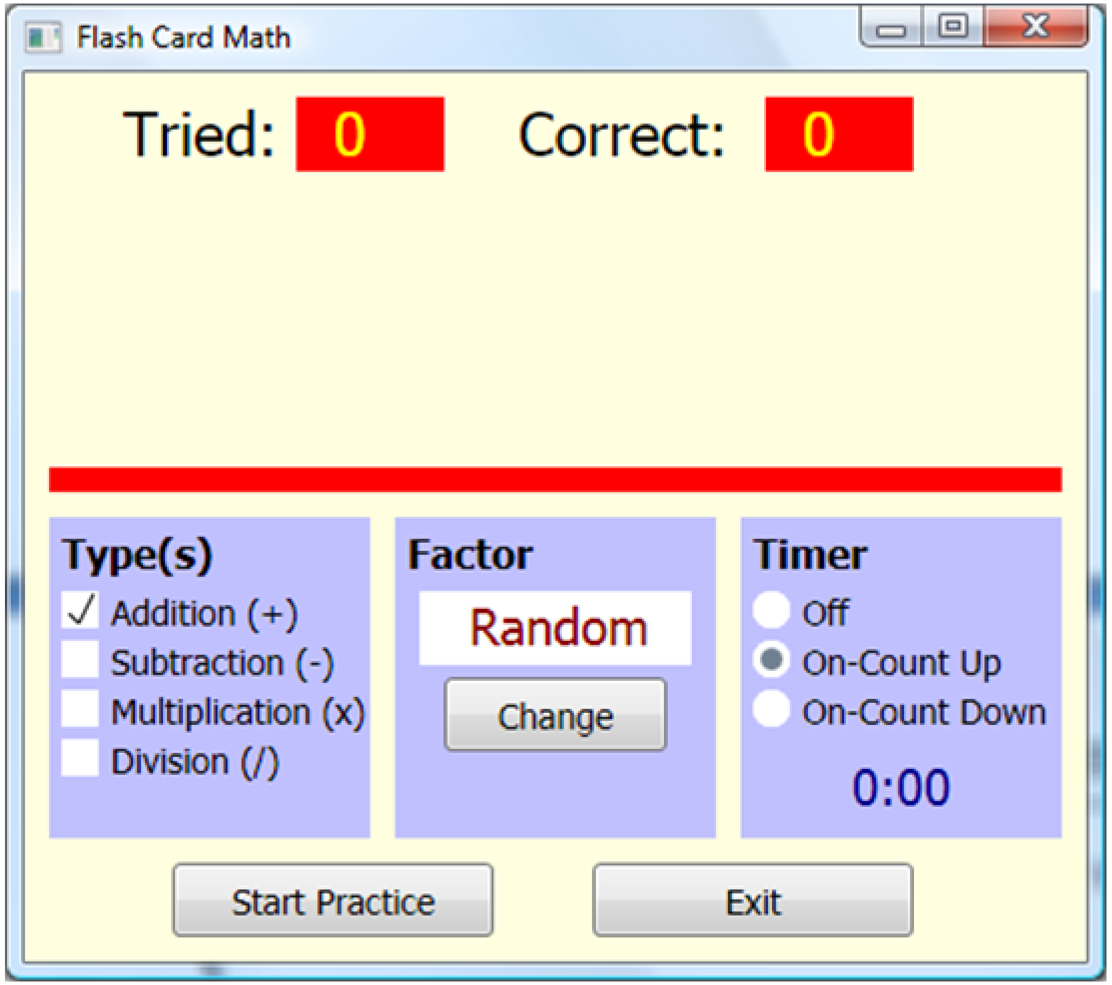
```
  EndSub
```

This subroutine uses another subroutine **FormatTime** that converts **ProblemTime** (seconds) as a **FormattedTime** in 00:00 format:

```
  Sub FormatTime
    Minutes = Math.Floor(ProblemTime / 60)
    Seconds = ProblemTime - Minutes * 60
    If (Seconds < 10) Then
      FormattedTime = Minutes + ":0" + Seconds
    Else
      FormattedTime = Minutes + ":" + Seconds
    EndIf
  EndSub
```

This subroutine takes the time (**ProblemTime**) in seconds and breaks it into minutes and seconds. Add this new code to your project.

Let's check both timer options to make sure the window changes as desired. **Save** and **Run** the program. Click **On-Count Up**. You should see:


Flash Card Math

Tried: 0 Correct: 0

**Type(s)**
- [✓] Addition (+)
- [ ] Subtraction (-)
- [ ] Multiplication (x)
- [ ] Division (/)

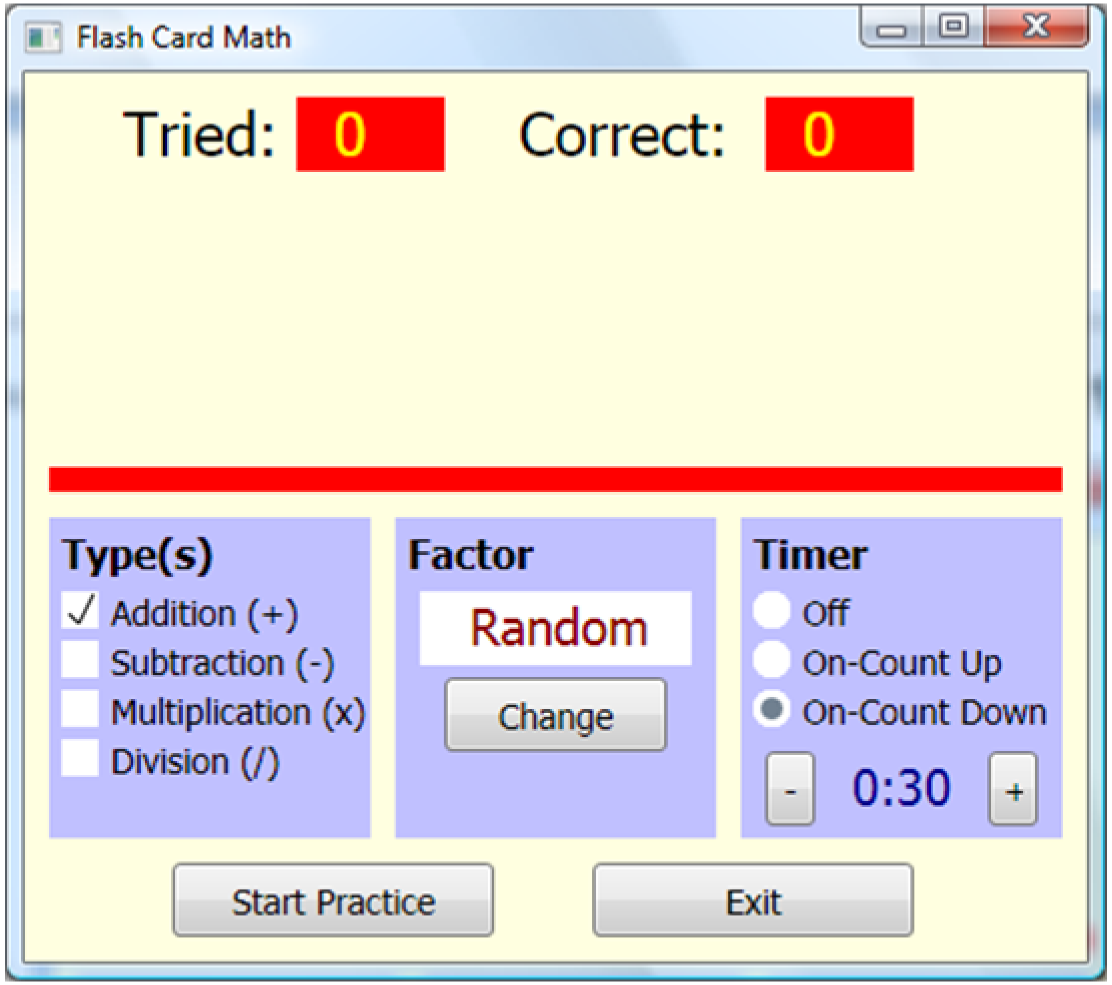
**Factor**
Random

**Timer**
- Off
- [✓ On-Count Up]
- On-Count Down

0:00

The ‘count-up’ time is displayed.

Now, choose **On-Count Down** to see:



Tried: 0 Correct: 0

✓ Addition (+)
☐ Subtraction (-)
☐ Multiplication (x)
☐ Division (/)

Random

Off
On-Count Up
[✓ On-Count Down]

0:30

The ‘count-down’ time is displayed along with adjustment button controls. In this mode, **TimerIndex** is used to initialize the **ProblemTime** variable. This value is changed by clicking on the +/- control buttons (**TimerPlusButton** and **TimerMinusButton**). We will keep **TimerIndex** between 1 and 60. This allows a maximum of 30 minutes (1800 seconds) for a timed flash card math session.

Add the shaded code to **ButtonClickedSub** to detect clicking on these buttons and adjustment of the **TimerIndex** value (along with the **TimeDisplay**):

```
Sub ButtonClickedSub

  B = Controls.LastClickedButton
  If (B = StartStopButton) Then
    StartStopButtonClicked()
  ElseIf (B = ExitButton) Then
    Program.End()
  ElseIf (B = FactorButton) Then
    FactorButtonClicked()
  ElseIf (B = TimerPlusButton) Then
    TimerIndex = TimerIndex + 1
    If (TimerIndex > 60) Then
      TimerIndex = 60
    EndIf
    ProblemTime = 30 * TimerIndex
    FormatTime()
    Shapes.SetText(TimeDisplay, FormattedTime)
  ElseIf (B = TimerMinusButton) Then
    TimerIndex = TimerIndex - 1
    If (TimerIndex < 1) Then
      TimerIndex = 1
    EndIf
    ProblemTime = 30 * TimerIndex
    FormatTime()
    Shapes.SetText(TimeDisplay, FormattedTime)
  EndIf
EndSub
```

A **Timer** object will be used to control the time display.  Add these at the end of **InitializeProgram** to set the **Interval** and establish the **Tick** event subroutine (**TimerTickSub**):

```
Timer.Interval = 1000

Timer.Tick = TimerTickSub

Timer.Pause()
```

Clicking **Start Practice** will start the timing process; the steps are:

- If **Off** (**TimerChoice = 0**) is selected, do nothing else.
- If **On-Count Up** (**TimerChoice = 1**) is selected:
    - Initialize **ProblemTime** to zero; display **ProblemTime**.
    - Start timer.
- If **On-Count Down** (**TimerChoice = 2**)  is selected:
    - Initialize **ProblemTime** to 30 times **TimerIndex**; display **ProblemTime**.
    - Hide **TimerPlusButton** and **TimerMinusButton**.
    - Start timer.

Clicking **Stop Practice** will stop the timing process.  The corresponding steps:

- Stop timer.
- Show **TimerPlusButton** and **TimerMinusButton** if **TimerChoice = 2**.

Each of these steps is handled in the **StartStopButton** subroutine.  The modified procedure (changes are shaded) is:

```
Sub StartStopButtonClicked

  If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then

    Controls.SetButtonCaption(StartStopButton, "Stop Practice")

    Controls.HideControl(ExitButton)

    NumberTried = 0

    NumberCorrect = 0

    Shapes.SetText(TriedDisplay, "0")

    Shapes.SetText(CorrectDisplay, "0")

    GetProblem()

    If (TimerChoice <> 0) Then
```

```
        If (TimerChoice = 1) Then
          ProblemTime = 0
        Else
          ProblemTime = 30 * TimerIndex
          Controls.HideControl(TimerMinusButton)
          Controls.HideControl(TimerPlusButton)
        EndIf
        FormatTime()
        Shapes.SetText(TimeDisplay, FormattedTime)
        Timer.Resume()
      EndIf
    Else
      Controls.SetButtonCaption(StartStopButton, "Start Practice")
      Controls.ShowControl(ExitButton)
      Shapes.SetText(ProblemDisplay, "")
      Timer.Pause()
      If (TimerChoice = 2) Then
        Controls.ShowControl(TimerMinusButton)
        Controls.ShowControl(TimerPlusButton)
      EndIf
    EndIf
  EndSub
```

Make the indicated changes.  We're almost ready to try the timing – just one more procedure to code.

When the timer is running, the time display (**TimeDisplay**) is updated every second (we use an **Interval** property of 1000).  The displayed time is incremented if counting up, decremented if counting down.  The steps involved for counting up are:

- Increment **ProblemTime** by 1.
- Display **ProblemTime**.
- If **ProblemTime** is 1800 (30 minutes), stop solving problems.

Note we limit the total solving time to 30 minutes.

The steps for counting down are:

- Decrement **ProblemTime** by 1.
- Display **ProblemTime**.
- If **ProblemTime** is 0, stop solving problems.

The code to update the displayed time is placed in the **TimerTickSub** subroutine. The code that implements the above steps are:

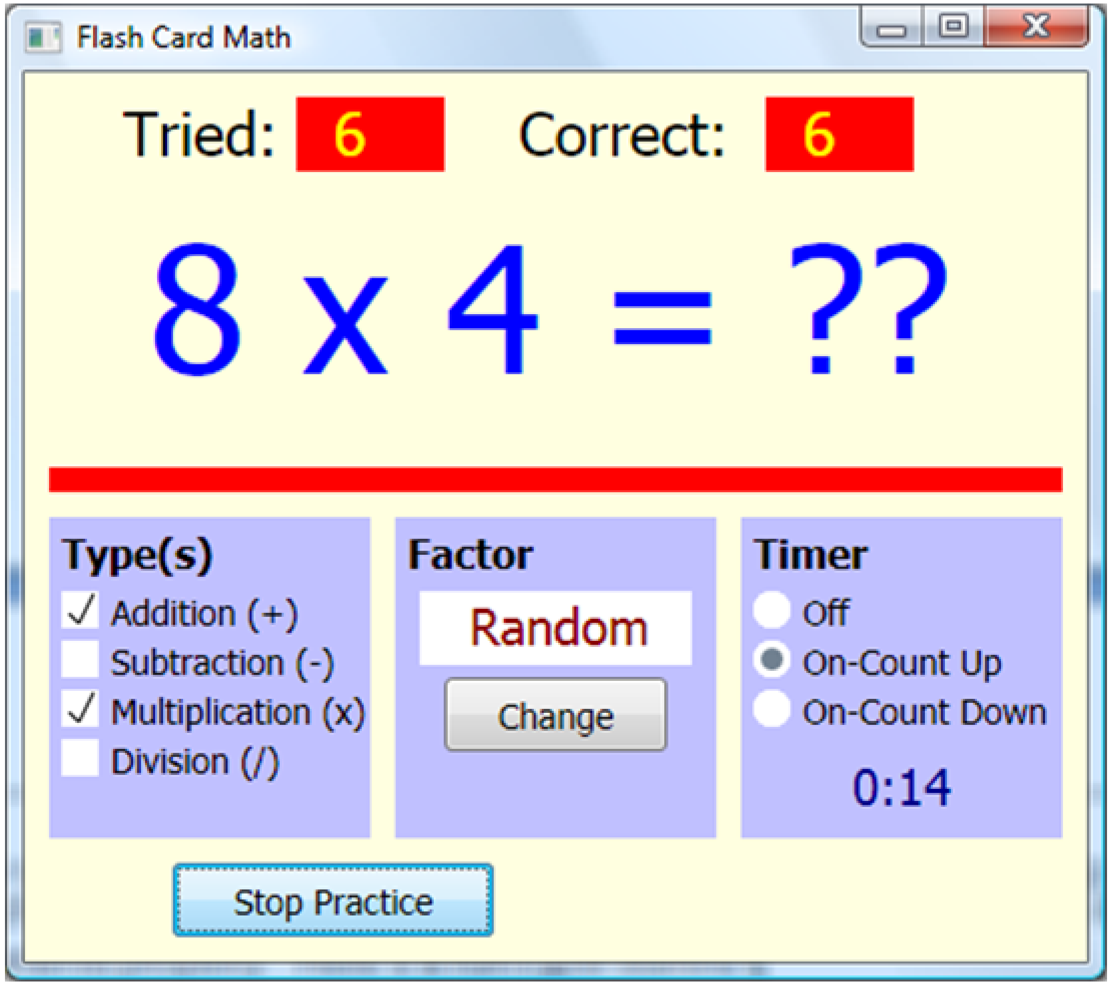
```
Sub TimerTickSub

  If (TimerChoice = 1) Then

    ProblemTime = ProblemTime + 1

    FormatTime()

    Shapes.SetText(TimeDisplay, FormattedTime)

    If ProblemTime >= 1800 Then

      StartStopButtonClicked()

    EndIf

  Else

    ProblemTime = ProblemTime - 1

    FormatTime()

    Shapes.SetText(TimeDisplay, FormattedTime)

    If (ProblemTime = 0) Then

      StartStopButtonClicked()

    EndIf

  EndIf
```

```
    EndSub
```

Notice to stop solving problems, we simulate a click on **Stop Practice** (the **StartStopButton** button). Add this procedure to the project.

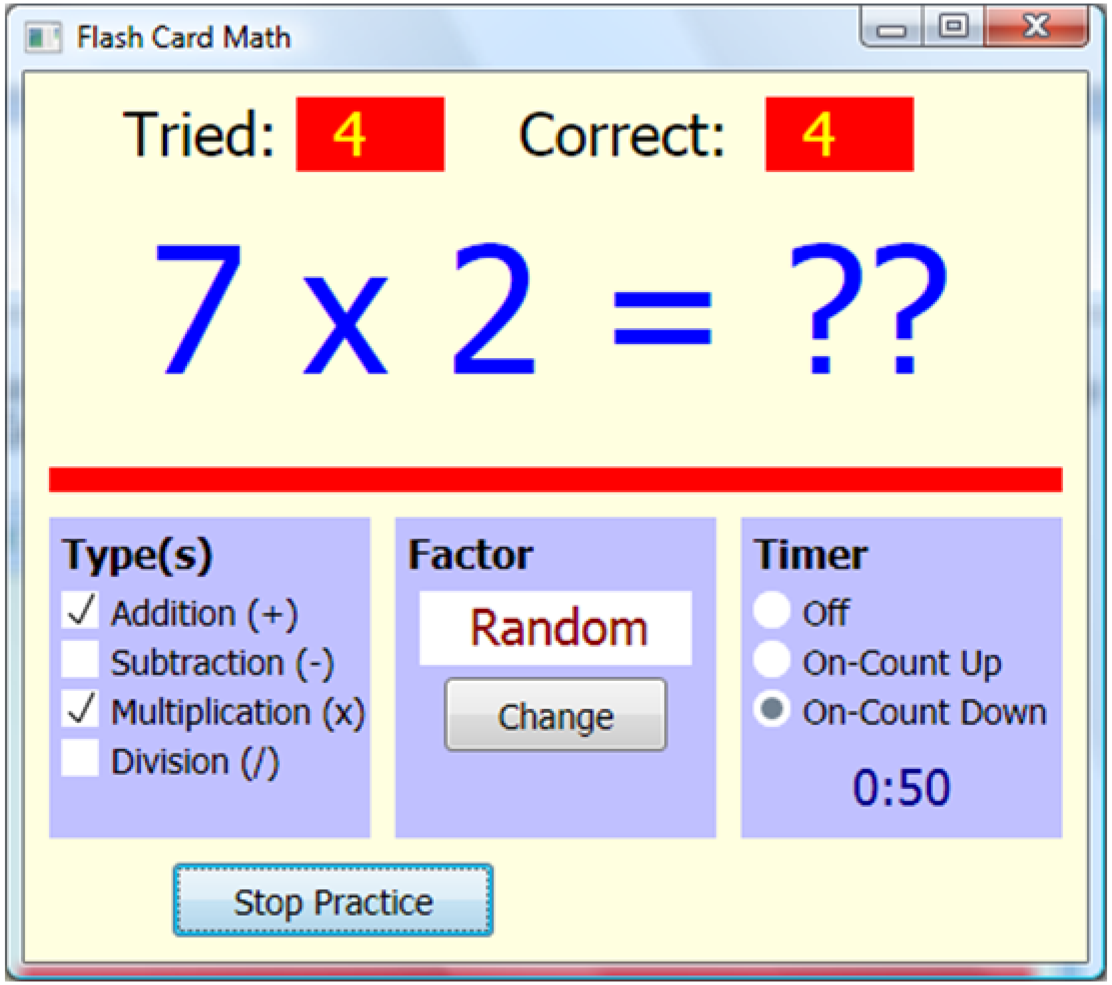
We're done implementing the modifications to add timing in the flash card math project. **Save** and **Run** the project. You want to make sure all the timer options work correctly. First, check to see that the project still works correctly with no timer.

Once you are convinced the no timer option still works, stop solving problems and choose the **On-Count Up** option. **Run** the project. Make sure the timer increments properly. Here's a run I just started:



Click **Stop Practice** at some point. You should also make sure the program automatically stops after 30 minutes (go have lunch while the program runs).

Choose the **On-Count Down** option. Change the amount of allowed time using the vertical scroll bar. Make sure it reaches a maximum of 30:00 (it has a minimum of 0:30). **Run** the project. Make sure the time decrements correctly. Here's a run I made using a starting time of 1:00:


Flash Card Math

Tried: 4 Correct: 4

7 x 2 = ??

Type(s)
- [✓] Addition (+)
- [ ] Subtraction (-)
- [✓] Multiplication (x)
- [ ] Division (/)

Factor
Random

Timer
- Off
- On-Count Up
- [✓ On-Count Down]

0:50

Make sure the program stops once the time elapses.

## Code Design – Presenting Results

Once a user stops solving problems, we want to let he/she know how well they did in answering problems. The information of use would be:

- The number of problems tried.
- The number of correct answers.
- The percentage score (**Score**).
- If timing, amount of elapsed time and time spent (on average) on each problem.

If timing up, the elapsed time is equal to **ProblemTime**. If timing down, the elapsed time is equal to the initial amount of time minus **ProblemTime**.

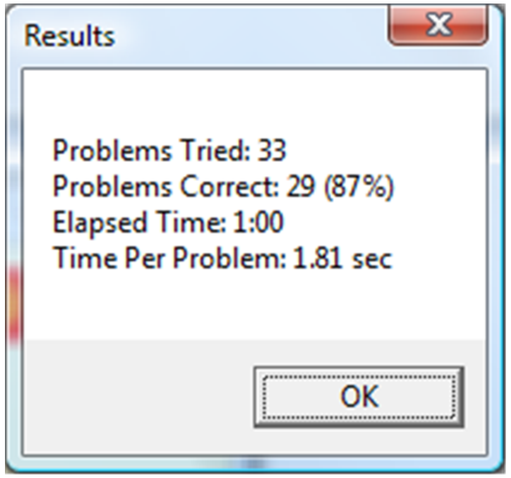
All of this information is readily available from the current variable set. The results are presented in the **StartStopButtonClicked** subroutine (following clicking of **Stop Practice**). We will use a simple message box to relay the results. The modified **StartStopButtonClicked** subroutine (changes are shaded) that displays the results is:

```
Sub StartStopButtonClicked

  If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then

    Controls.SetButtonCaption(StartStopButton, "Stop Practice")

    Controls.HideControl(ExitButton)

    NumberTried = 0

    NumberCorrect = 0

    Shapes.SetText(TriedDisplay, "0")

    Shapes.SetText(CorrectDisplay, "0")

    GetProblem()

    If (TimerChoice <> 0) Then

      If (TimerChoice = 1) Then

        ProblemTime = 0

      Else

        ProblemTime = 30 * TimerIndex

        Controls.HideControl(TimerMinusButton)
```

```
          Controls.HideControl(TimerPlusButton)

        EndIf

        FormatTime()

        Shapes.SetText(TimeDisplay, FormattedTime)

        Timer.Resume()

      EndIf

    Else

      Controls.SetButtonCaption(StartStopButton, "Start Practice")

      Controls.ShowControl(ExitButton)

      Shapes.SetText(ProblemDisplay, "")

      Timer.Pause()

      If (TimerChoice = 2) Then

        Controls.ShowControl(TimerMinusButton)

        Controls.ShowControl(TimerPlusButton)

      EndIf

      If (NumberTried > 0) Then

        CRLF = Text.GetCharacter(13)

        Score = Math.Floor(100 * (NumberCorrect / NumberTried))

        Message = "Problems Tried: " + NumberTried + CRLF

        Message = Message + "Problems Correct: " + NumberCorrect + " (" + Score + "%)"
  + CRLF

        If (TimerChoice = 0) Then

          Message = Message + "Timer Off"

        Else

          If (TimerChoice = 2) Then

            ProblemTime = 30 * TimerIndex - ProblemTime
          EndIf

          FormatTime()

          Message = Message + "Elapsed Time: " + FormattedTime + CRLF

          Message = Message + "Time Per Problem: " + Math.Floor(100 * (ProblemTime /
  NumberTried)) / 100 + " sec"

        EndIf

        GraphicsWindow.ShowMessage(Message, "Results")

      EndIf

    EndIf

  EndSub
```

Add the noted changes.

One last time – **Save** and **Run** the project. Solve some problems and see the results. Make sure the results display correctly whether timing or not. Here is a set of results I received while using the timing down option:



```
Problems Tried: 33
Problems Correct: 29 (87%)
Elapsed Time: 1:00
Time Per Problem: 1.81 sec
```

## Flash Card Math Quiz Project Code Listing

Here is the complete listing of the **Flash Card Math Quiz** Small Basic program:

```
' Flash Card Math
```

```
InitializeProgram()

Sub InitializeProgram

  'graphics window

  GraphicsWindow.Width = 430

  GraphicsWindow.Height = 360

  GraphicsWindow.Title = "Flash Card Math"

  GraphicsWindow.BackgroundColor = "LightYellow"

  'labels/scores

  GraphicsWindow.BrushColor = "Black"

  GraphicsWindow.FontBold = "false"

  GraphicsWindow.FontSize = 24

  GraphicsWindow.DrawText(40, 10, "Tried:")

  GraphicsWindow.DrawText(200, 10, "Correct:")

  GraphicsWindow.BrushColor = "Red"

  GraphicsWindow.FillRectangle(110, 10, 60, 30)

  GraphicsWindow.FillRectangle(300, 10, 60, 30)

  GraphicsWindow.BrushColor = "Yellow"

  TriedDisplay = Shapes.AddText("0")

  Shapes.Move(TriedDisplay, 125, 10)

  CorrectDisplay = Shapes.AddText("0")

  Shapes.Move(CorrectDisplay, 315, 10)

  'problem display

  GraphicsWindow.BrushColor = "Blue"

  GraphicsWindow.FontSize = 72

  ProblemDisplay = Shapes.AddText("")
```

```
Shapes.Move(ProblemDisplay, 50, 50)

'divider

GraphicsWindow.BrushColor = "Red"

GraphicsWindow.FillRectangle(10, 160, 410, 10)

'problem types/factor/timer

GraphicsWindow.BrushColor = GraphicsWindow.GetColorFromRGB(192, 192, 255)

GraphicsWindow.FillRectangle(10, 180, 130, 130)

GraphicsWindow.FillRectangle(150, 180, 130, 130)

GraphicsWindow.FillRectangle(290, 180, 130, 130)

GraphicsWindow.BrushColor = "Black"

GraphicsWindow.FontSize = 16

GraphicsWindow.FontBold = "true"

GraphicsWindow.DrawText(15, 185, "Type(s)")

GraphicsWindow.DrawText(155, 185, "Factor")

GraphicsWindow.DrawText(295, 185, "Timer")

'problem types

GraphicsWindow.BrushColor = "Black"

GraphicsWindow.FontSize = 14

GraphicsWindow.FontBold = "false"

GraphicsWindow.DrawText(35, 210, "Addition (+)")

GraphicsWindow.DrawText(35, 230, "Subtraction (-)")

GraphicsWindow.DrawText(35, 250, "Multiplication (x)")

GraphicsWindow.DrawText(35, 270, "Division (/)")

GraphicsWindow.BrushColor = "White"

GraphicsWindow.FillRectangle(15, 210, 15, 15)
GraphicsWindow.FillRectangle(15, 230, 15, 15)

GraphicsWindow.FillRectangle(15, 250, 15, 15)

GraphicsWindow.FillRectangle(15, 270, 15, 15)

ProblemSelected[1] = "true"

ProblemSelected[2] = "false"

ProblemSelected[3] = "false"

ProblemSelected[4] = "false"

Selections = 1

ProblemTypeClicked = 1

MarkProblemType()

'factor

GraphicsWindow.BrushColor = "White"

GraphicsWindow.FillRectangle(160, 210, 110, 30)

GraphicsWindow.BrushColor = "DarkRed"

GraphicsWindow.FontSize = 20

FactorDisplay = Shapes.AddText("Random")

Shapes.Move(FactorDisplay, 180, 212)

FactorChoice = -1

GraphicsWindow.BrushColor = "Black"

GraphicsWindow.FontSize = 14

FactorButton = Controls.AddButton("Change", 170, 245)

Controls.SetSize(FactorButton, 90, 30)

'timer choices

GraphicsWindow.BrushColor = "Black"

GraphicsWindow.FontSize = 14
GraphicsWindow.FontBold = "false"

GraphicsWindow.DrawText(315, 210, "Off")

GraphicsWindow.DrawText(315, 230, "On-Count Up")

GraphicsWindow.DrawText(315, 250, "On-Count Down")

GraphicsWindow.BrushColor = "White"

GraphicsWindow.FillEllipse(295, 210, 15, 15)

GraphicsWindow.FillEllipse(295, 230, 15, 15)

GraphicsWindow.FillEllipse(295, 250, 15, 15)

TimerChoice = 0 ' 0-off, 1 -on/up, 2- on/down

MarkTimerChoice()

TimerIndex = 1

GraphicsWindow.BrushColor = "DarkBlue"

GraphicsWindow.FontSize = 20

TimeDisplay = Shapes.AddText("0:30")

Shapes.Move(TimeDisplay, 335, 277)

GraphicsWindow.BrushColor = "Black"

GraphicsWindow.FontSize = 14

TimerPlusButton = Controls.AddButton("+", 390, 275)

Controls.SetSize(TimerPlusButton, 20, 30)

TimerMinusButton = Controls.AddButton("-", 300, 275)

Controls.SetSize(TimerMinusButton, 20, 30)

Shapes.HideShape(TimeDisplay)

Controls.HideControl(TimerMinusButton)

Controls.HideControl(TimerPlusButton)

'buttons
```

```
  GraphicsWindow.BrushColor = "Black"
  GraphicsWindow.FontSize = 14
  StartStopButton = Controls.AddButton("Start Practice", 60, 320)
  Controls.SetSize(StartStopButton, 130, 30)
  ExitButton = Controls.AddButton("Exit", 230, 320)
  Controls.SetSize(ExitButton, 130, 30)
  Controls.ButtonClicked = ButtonClickedSub
  GraphicsWindow.KeyDown = KeyDownSub
  GraphicsWindow.MouseDown = MouseDownSub
  Timer.Interval = 1000
  Timer.Tick = TimerTickSub
  Timer.Pause()
EndSub
Sub ButtonClickedSub
  B = Controls.LastClickedButton
  If (B = StartStopButton) Then
    StartStopButtonClicked()
  ElseIf (B = ExitButton) Then
    Program.End()
  ElseIf (B = FactorButton) Then
    FactorButtonClicked()
  ElseIf (B = TimerPlusButton) Then
    TimerIndex = TimerIndex + 1
    If (TimerIndex > 60) Then
      TimerIndex = 60
    EndIf
    ProblemTime = 30 * TimerIndex
    FormatTime()
    Shapes.SetText(TimeDisplay, FormattedTime)
  ElseIf (B = TimerMinusButton) Then
    TimerIndex = TimerIndex - 1
    If (TimerIndex < 1) Then
      TimerIndex = 1
    EndIf
    ProblemTime = 30 * TimerIndex
    FormatTime()
    Shapes.SetText(TimeDisplay, FormattedTime)
  EndIf
EndSub

Sub StartStopButtonClicked
  If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then
    Controls.SetButtonCaption(StartStopButton, "Stop Practice")
    Controls.HideControl(ExitButton)
    NumberTried = 0
    NumberCorrect = 0
    Shapes.SetText(TriedDisplay, "0")
    Shapes.SetText(CorrectDisplay, "0")
    GetProblem()
    If (TimerChoice <> 0) Then
      If (TimerChoice = 1) Then
        ProblemTime = 0
      Else
        ProblemTime = 30 * TimerIndex
        Controls.HideControl(TimerMinusButton)
        Controls.HideControl(TimerPlusButton)
      EndIf
      FormatTime()
      Shapes.SetText(TimeDisplay, FormattedTime)
      Timer.Resume()
    EndIf
  Else
    Controls.SetButtonCaption(StartStopButton, "Start Practice")
    Controls.ShowControl(ExitButton)
    Shapes.SetText(ProblemDisplay, "")
    Timer.Pause()
    If (TimerChoice = 2) Then
      Controls.ShowControl(TimerMinusButton)
      Controls.ShowControl(TimerPlusButton)
    EndIf
    If (NumberTried > 0) Then
      CRLF = Text.GetCharacter(13)
      Score = Math.Floor(100 * (NumberCorrect / NumberTried))
      Message = "Problems Tried: " + NumberTried + CRLF
      Message = Message + "Problems Correct: " + NumberCorrect + " (" + Score + "%)" +
CRLF
```

```
      If (TimerChoice = 0) Then
        Message = Message + "Timer Off"
      Else
        If (TimerChoice = 2) Then
          ProblemTime = 30 * TimerIndex - ProblemTime
        EndIf
        FormatTime()
        Message = Message + "Elapsed Time: " + FormattedTime + CRLF
        Message = Message + "Time Per Problem: " + Math.Floor(100 * (ProblemTime /
NumberTried)) / 100 + " sec"
      EndIf
      GraphicsWindow.ShowMessage(Message, "Results")
    EndIf
  EndIf
EndSub
Sub GetProblem
  ProblemType = 0
  While (ProblemType = 0)
    P = Math.GetRandomNumber(4)
    If (P = 1 And ProblemSelected[1]) Then
      'Addition
      ProblemType = P
      Number = Math.GetRandomNumber(10) - 1
      GetFactor()
      CorrectAnswer = Number + Factor
      Problem = Number + " + " + Factor + " = "
    ElseIf (P = 2 And ProblemSelected[2]) Then
      'Subtraction
      ProblemType = P
      GetFactor()
      CorrectAnswer = Math.GetRandomNumber(10) - 1
      Number = CorrectAnswer + Factor
      Problem = Number + " - " + Factor + " = "
    ElseIf (P = 3 And ProblemSelected[3]) Then
      'Multiplication
      ProblemType = P
      Number = Math.GetRandomNumber(10) - 1
      GetFactor()
      CorrectAnswer = Number * Factor
      Problem = Number + " x " + Factor + " = "
    ElseIf (P = 4 And ProblemSelected[4]) Then
      'Division
      ProblemType = P
      GetFactor()
      CorrectAnswer = Math.GetRandomNumber(10) - 1
      Number = CorrectAnswer * Factor
      Problem = Number + " / " + Factor + " = "
    EndIf
  EndWhile
  YourAnswer = ""
  DigitNumber = 1
  If (CorrectAnswer < 10) Then

    NumberDigits = 1

    Shapes.SetText(ProblemDisplay, Problem + "?")

  Else

    NumberDigits = 2

    Shapes.SetText(ProblemDisplay, Problem + "??")

  EndIf

EndSub

Sub GetFactor

  If (FactorChoice = -1) Then

    If (ProblemType = 4) Then

      Factor = Math.GetRandomNumber(9)

    Else

      Factor = Math.GetRandomNumber(10) - 1

    EndIf

  Else

    Factor = FactorChoice

  EndIf

EndSub


Sub KeyDownSub

  If (Controls.GetButtonCaption(StartStopButton) = "Stop Practice") Then

    'can only check keystrokes when practicing-only allow number keys

    'number is last character in keypressed

    KeyPressed = Text.GetSubTextToEnd(GraphicsWindow.LastKey,
Text.GetLength(GraphicsWindow.LastKey))
```

```
    If (Text.GetCharacterCode(KeyPressed) >= Text.GetCharacterCode("0") And
Text.GetCharacterCode(KeyPressed) <= Text.GetCharacterCode("9")) Then

      YourAnswer = Text.Append(YourAnswer, KeyPressed)

      If DigitNumber <> NumberDigits Then

        DigitNumber = DigitNumber + 1

        Shapes.SetText(ProblemDisplay, Problem + YourAnswer + "?")

      Else

        NumberTried = NumberTried + 1

        'check answer

        If (YourAnswer = CorrectAnswer) Then

          NumberCorrect = NumberCorrect + 1

        EndIf

        Shapes.SetText(TriedDisplay, NumberTried)

        Shapes.SetText(CorrectDisplay, NumberCorrect)

        GetProblem()

      EndIf

    EndIf

  EndIf

EndSub


Sub MouseDownSub

  X = GraphicsWindow.MouseX

  Y = GraphicsWindow.MouseY

  'problem selections - must always have at least one selected

  If (X > 15 And X < 30 And Y > 210 And Y < 285) Then

    If (Y > 210 And Y < 225) Then
     'clicked addition
     ProblemTypeClicked = 1
    ElseIf (Y > 230 And Y < 245) Then
      'clicked subtraction
      ProblemTypeClicked = 2
    ElseIf (Y > 250 And Y < 265) Then
      'clicked multiplication
     ProblemTypeClicked = 3
    ElseIf (Y > 270 And Y < 285) Then
     'clicked division
      ProblemTypeClicked = 4
    EndIf
    If (ProblemSelected[ProblemTypeClicked] And Selections <> 1) Then
      'clear choice if not last one selected
      Selections = Selections - 1
      ProblemSelected[ProblemTypeClicked] = "false"
      ProblemTypeClicked = -ProblemTypeClicked
      MarkProblemType()
    ElseIf (ProblemSelected[ProblemTypeClicked] = "false") Then
      'mark choice
      Selections = Selections + 1
      ProblemSelected[ProblemTypeClicked] = "true"
      MarkProblemType()
      'make sure zero not selected factor if division selected
      If (ProblemTypeClicked = 4 And FactorChoice = 0) Then
```

```
      FactorButtonClicked()
    EndIf
  EndIf
EndIf
If (Controls.GetButtonCaption(StartStopButton) = "Start Practice") Then
  'timer selections - only one can be selected
  If (X > 295 And X < 310 And Y > 210 And Y < 265) Then
    'problem selections - must always have at least one selected
    If (Y > 210 And Y < 225) Then
      'clicked timer off
      TimerChoice = 0
      MarkTimerChoice()
      Shapes.HideShape(TimeDisplay)
      Controls.HideControl(TimerMinusButton)
      Controls.HideControl(TimerPlusButton)
    ElseIf (Y > 230 And Y < 245) Then
      'clicked timer on - count up
      TimerChoice = 1
      MarkTimerChoice()
      Shapes.ShowShape(TimeDisplay)
      Controls.HideControl(TimerMinusButton)
      Controls.HideControl(TimerPlusButton)
      ProblemTime = 0
      FormatTime()
      Shapes.SetText(TimeDisplay, FormattedTime)
      ElseIf (Y > 250 And Y < 265) Then
        'clicked timer on - count down
        TimerChoice = 2
        MarkTimerChoice()
        Shapes.ShowShape(TimeDisplay)
        Controls.ShowControl(TimerMinusButton)
        Controls.ShowControl(TimerPlusButton)
        ProblemTime = 30 * TimerIndex
        FormatTime()
        Shapes.SetText(TimeDisplay, FormattedTime)
      EndIf
    EndIf
  EndIf
EndSub
Sub MarkProblemType
  If (ProblemTypeClicked < 0) Then
    ProblemTypeClicked = Math.Abs(ProblemTypeClicked)
    GraphicsWindow.BrushColor = "White"
    GraphicsWindow.FillRectangle(15, 190 + ProblemTypeClicked * 20, 15, 15)
  Else
    GraphicsWindow.PenColor = "Black"
    GraphicsWindow.PenWidth = 1
    GraphicsWindow.DrawLine(18, 199 + ProblemTypeClicked * 20, 22, 203 +
ProblemTypeClicked * 20)
    GraphicsWindow.DrawLine(22, 203 + ProblemTypeClicked * 20, 28, 191 +
ProblemTypeClicked * 20)
```

```
  EndIf
EndSub

Sub FactorButtonClicked
  'change factor choice
  FactorChoice = FactorChoice + 1
  If (FactorChoice > 9) Then
    FactorChoice = -1
    Shapes.SetText(FactorDisplay, "Random")
  Else
    If (ProblemSelected[4] And FactorChoice = 0) Then
      'no zero if doing division
      FactorChoice = 1
    EndIf
    Shapes.SetText(FactorDisplay, Text.Append("     ", FactorChoice))
  EndIf
EndSub
Sub MarkTimerChoice
  GraphicsWindow.BrushColor = "White"
  GraphicsWindow.FillEllipse(295, 210, 15, 15)
  GraphicsWindow.FillEllipse(295, 230, 15, 15)
  GraphicsWindow.FillEllipse(295, 250, 15, 15)
  GraphicsWindow.BrushColor = "SlateGray"
  GraphicsWindow.FillEllipse(298, 213 + TimerChoice * 20, 9, 9)
EndSub
```

```
Sub FormatTime
  Minutes = Math.Floor(ProblemTime / 60)
  Seconds = ProblemTime - Minutes * 60
  If (Seconds < 10) Then
    FormattedTime = Minutes + ":0" + Seconds
  Else
    FormattedTime = Minutes + ":" + Seconds
  EndIf
EndSub

Sub TimerTickSub
  If (TimerChoice = 1) Then
    ProblemTime = ProblemTime + 1
    FormatTime()
    Shapes.SetText(TimeDisplay, FormattedTime)
    If ProblemTime >= 1800 Then
      StartStopButtonClicked()
    EndIf
  Else
    ProblemTime = ProblemTime - 1
    FormatTime()
    Shapes.SetText(TimeDisplay, FormattedTime)
    If (ProblemTime = 0) Then
      StartStopButtonClicked()
    EndIf
  EndIf
EndSub
```

## Flash Card Math Quiz Project Review

The **Flash Card Math Quiz** project is now complete. **Save** and **Run** the project and make sure it works as designed. Recheck that all options work and interact properly. Let your kids (or anyone else) have fun tuning up their basic math skills.

If there are errors in your implementation, go back over the steps of window and code design. Use the debugger when needed. Go over the developed code – make sure you understand how different parts of the project were coded. As mentioned in the beginning of this chapter, the completed project is saved as **FlashCard** in the **HomeSB\HomeSB Projects\FlashCard** folder.

While completing this project, new concepts and skills you should have gained include:

- How to make selections among options.
- Use of the KeyDown event for input.
- Using a message box to report results.

## Flash Card Math Quiz Project Enhancements

Possible enhancements to the flash card math project include:

- As implemented, the only feedback a user gets about entered answers is an update of the score. Some kind of audible feedback would be a big improvement (a positive sound for correct answer, a negative sound for a wrong answer). We discuss adding sounds to a project in the final chapter – you might like to look ahead.
- When a user stops answering problems, it would be nice to have a review mode where the problems missed are presented. You would need some way to save each problem that was answered incorrectly.
- Kids like rewards. As you gain more programming skills, a nice visual display of some sort for good work would be a fun addition.
- Currently, once a problem is answered, the next problem is immediately displayed. Some kind of delay (perhaps make it optional and adjustable) might be desired.