Documentation

What is ICE?

ICE Compiler is a program that compiles TI-BASIC-like language to the faster machine code, assembly. Since the calculator directly reads assembly, it doesn't need to compile TI-BASIC, and run that assembly code, thus compiled programs would be much faster. In this version, v1.5, I've added a bunch of functions to create the best graphics for your games, with a reasonable speed. There are also more math functions added, to make more possible. Special in version 1.5 is, that you can now interact with BASIC, using "SetBASICVar(", "GetBASICVar(" and "prgmPROGRAM"!

Limitations

All the numbers and variables are unsigned 3-byte integers, which means you can use any number between 0 and 16777215. If you go above that number, it takes your number modulo 16777215, but I think this should be enough to create the best programs.

Installment

If you don't use CEmu

1) Connect your calc to your computer/laptop with TI-Connect CE and a cable
2) Transfer program ICE to your calculator

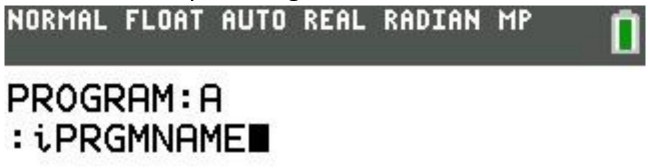
If you use CEmu

1) Open CEmu
2) Drag and drop program ICE to the screen of CEmu

You are now ready to use it.

Build my first program

ICE only recognizes ICE programs with a special header. Each ICE program should have the first token the imaginary *i*, and then the output name in capital letters. If ICE encounters a token which is not a letter, it will stop reading the name, and starts with compiling. This is an example:



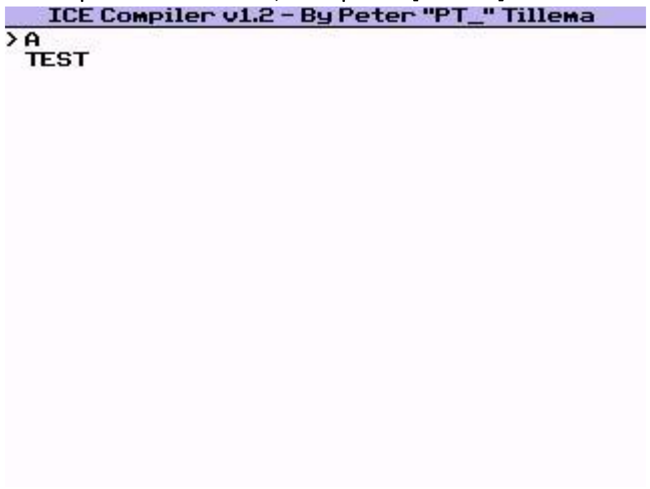
```
NORMAL FLOAT AUTO REAL RADIAN MP

PROGRAM:A
:iPRGMNAME
```

Compiling

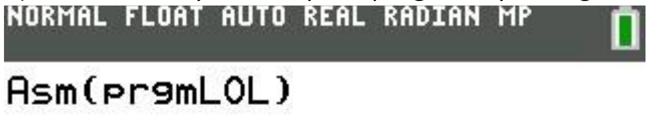
1) Be sure you have a program to be compiled
2) Select the token `Asm(` from the Catalog. Then press [PRGM] and select `ICE`. You should now see this on the homescreen: `Asm(prgmICE`. Press [ENTER] to start.

3) If your first token was the imaginary *i*, it will show you the name. Select the program you want to compile from the list, and press [ENTER].



```
ICE Compiler v1.2 - By Peter "PT_" Tillema
>A
 TEST
```

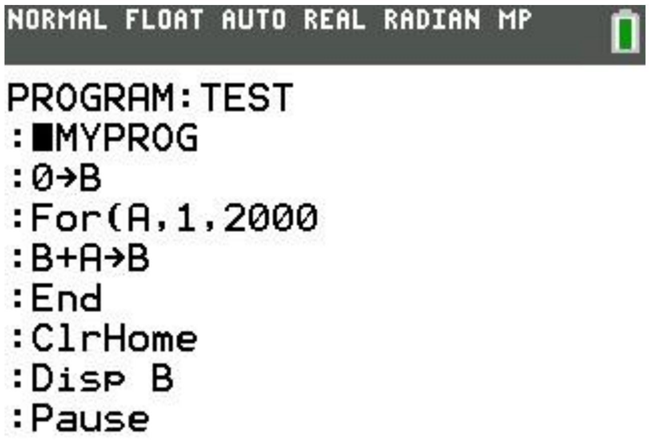
4) ICE will now start with compiling. This is almost instant, so you shouldn't see the status bar (but it is there). Whenever something went wrong during compiling, you see a message with a short explanation, and the line number where that error occurred. You should be able to find the error yourself, and if not, please don't hesitate to contact me (details at the end)
5) If no error occurred, you see the message that it is successfully compiled.
6) Press any key to quit ICE. If you had already a program with the output name, it will be overwritten (the output name can't be the same as the source name)
7) Now can run your compiled program by calling `Asm(prgm<programname>)`



```
NORMAL FLOAT AUTO REAL RADIAN MP
Asm(prgmLOL)
```

Let's start!

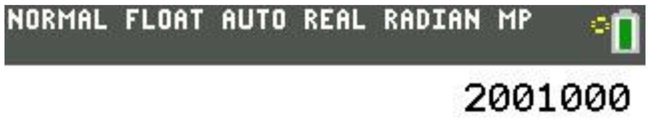
The most important step is to backup your calculator! Despite that I've tested it very well, I can't promise that no RAM clear will happen. I highly recommend to use CEmu, that is the safest way to test things. Well, let's hope you get no RAM clear, and you want to compile your first program. Math in ICE programs is pretty straightforward, since it looks like the BASIC math. The order of operations does matter, and you can use anywhere parenthesis. *Note: ^ is not implemented*. But there is more stuff! You can use loops, math functions, graph functions. Let me give you an example. You want to know what 1+2+3+…+2000 is? Let's do it!



```
NORMAL FLOAT AUTO REAL RADIAN MP

PROGRAM:TEST
:■MYPROG
:0→B
:For(A,1,2000
:B+A→B
:End
:ClrHome
:Disp B
:Pause
```

You don't understand this code? It's very easy! The sum of the first 2000 integers starts with 0, so `0->B`. Now, we use a For-loop to get the sum. Each time in the For-loop, the integer A will be added to B, so if A goes to 2000, B equals 1+2+3+…+2000 ! To make it more clear we got the right result, we clear the homescreen, display the sum, and wait for the user to press [ENTER], otherwise it will be too fast and we can't see the result.

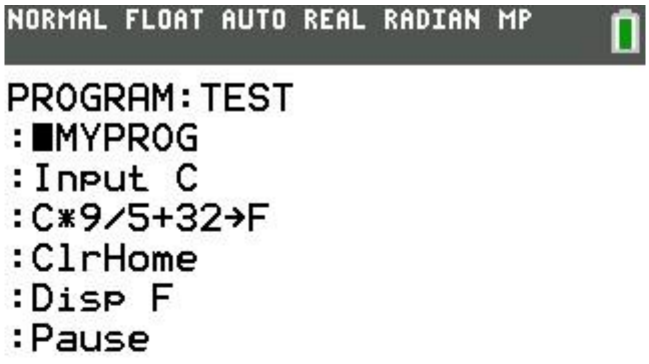


```
NORMAL FLOAT AUTO REAL RADIAN MP
                                2001000
```

Is the outcome right? YES!

So, you have created your first ICE program successfully, without a crash, and it seems it worked. Let's dive into more complicated features in ICE programs!

Let's say we want the user to input a variable! That is definitely possible, just use `Input <var>`. Very simple, right? Let's make a simple program that asks for the temperature in degrees, and converts that to Fahrenheit. The formula is `T(F)=T(C)*9/5+32`. Now that we are going to use division, I need to tell you that ICE can only handle integers, not floating point numbers. So `T(C)*9/5` will always be rounded down. Let's try it.



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:TEST
:■MYPROG
:Input C
:C*9/5+32→F
:ClrHome
:Disp F
:Pause
```

This code asks for a temperature, and automatically stores that in variable C. Then, according to the formula, F=C*9/5+32. Then we display the temperature in Fahrenheit to check it.

One common mistake
This formula is fine, but one can also take C/5*9+32. DON'T DO THIS!! Remember what I just said, division will be rounded down. If we divide by 5, it will be rounded down and then multiply it by 9, but we can get the wrong results! Try it yourself, and test both programs with a temperature of **9** degrees. The right formula gives us 48 Fahrenheit, but the last program only 41… We can explain this mathematically:

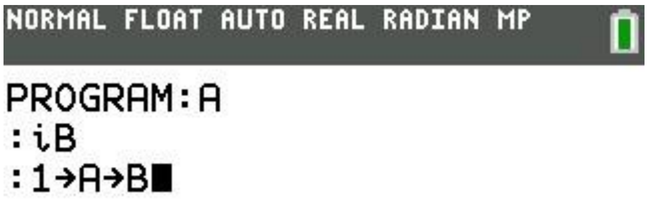
- 9*9=81/5 (rounded down)=16+32=48
- 9/5 (rounded down)=1*9=9+32=41

You notice the difference? That is why it is ALWAYS useful to multiply first, before division.

There is much, much more possible. I advise you to take a look at `commands.html` for a list of the available tokens and functions. You see some functions that are not accessible in the OS. When you press [TRACE] while editing a BASIC program, you can a list with all the 'weird' tokens and all the graph functions. You can select one (there are multiple tabs, press [RIGHT] or [LEFT] to get more functions), and when you press [ENTER], that function will be inserted into the program editor. You can also learn more by looking at the included examples, I try to handle as much functions as possible.

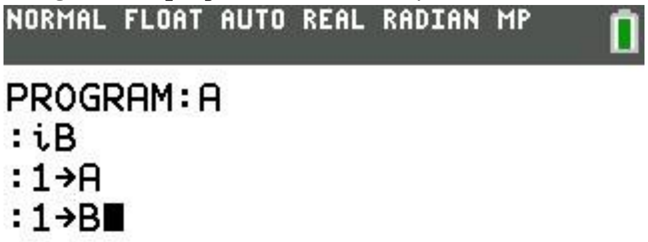
Advanced usage

Luckily there is more possible in ICE. One thing you need to remember, is that you can chain → operators. For example "1->A->B" is definitely possible! This saves much spaces and time, so I highly recommend you to use it! Let's see the difference, by making this program:



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:A
:iB
:1→A→B
```

This gives me `prgmB` a size of 28 bytes. Let's see what the size is when you didn't do that.



```
NORMAL FLOAT AUTO REAL RADIAN MP
PROGRAM:A
:iB
:1→A
:1→B
```

32 bytes! We can just save 4 unnecessary bytes by combining the →operators!

Special warning: it can happen that ICE encounters an unknown bug, you will see that message at the screen. **Please report this always to me, if possible send me the source code so I can check what went wrong.** It's not useful, if you only say there is a huge bug, but you won't give me access to your source code, no one has benefit with that.

Thanks to everyone who contributed, and I hope you have a lot of fun with it!

Peter Tillema

Contact me at petertillema@solcon.nl or look at http://ice.cemetech.net for more information about ICE, and some Tips and Tricks!
