CoCoA-5 - Feature #500

Interpreter: is it possible to avoid useless prompts?

28 Mar 2014 09:24 - Anna Maria Bigatti

Status:	Closed	**Start date:**	28 Mar 2014
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	Parser/Interpreter	**Estimated time:**	7.47 hours
Target version:	CoCoA-5.4.0	**Spent time:**	7.45 hours

Description

If we input

```
1;
2;
3;
```

we get 3 prompts.
Is it possible to check, before printing the prompt, there is already new input in execution?
So that the prompt is printed only when it is **really waiting for input**?

Maybe not, but it would be nice ...

Related issues:

Related to CoCoA-5 - Feature #352: Should SourceRegion echo the "region"?	**Closed**	**21 May 2013**
Related to CoCoA-5 - Feature #133: Qt GUI: Make better distinction between in...	**New**	**19 Apr 2012**
Related to CoCoA-5 - Support #251: How to add a test for CoCoA-5 (CoCoAInterp...	**Closed**	**02 Oct 2012**
Related to CoCoA-5 - Bug #697: Interpreter: Avoid outputting an empty line af...	**Closed**	**11 May 2015**
Related to CoCoA-5 - Support #911: CoCoA-5 prompt for incomplete input	**Closed**	**27 Jul 2016**

History

#1 - 28 Mar 2014 15:14 - John Abbott

JAA thinks that this could be hard to do portably.

A potentially simpler solution would be to have some way of marking "end-of-block-of-input" (some sort of invisible character that has the same semantics as newline).

#2 - 09 Apr 2014 17:31 - John Abbott

- Target version changed from CoCoA-5.1.0 Easter14 to CoCoA-5.1.1 Seoul14

#3 - 29 Aug 2014 11:58 - John Abbott

This appears to be impossible in C++ (version 2003), at least in a fully portable way.

I have found a post which explains how to do it if you compile with g++: see http://www.cplusplus.com/forum/general/74355/

The solution proposed there does work for me (I compiled with Apple's g++ version 4.2.1), but we cannot accept it as it depends on compiling with g++ -- I could add a CPP #ifdef but then the expected output of the CoCoA-5 tests would depend on which compiler was used to compile CoCoA-5.

Attempts to use the in_avail member function failed because it always returns 0; this is allowed by the definition. Even trying to use the readsome member function produced the same overall effect (since readsome is allowed to return without reading anything).

Altogether a frustrating two hour waste of time. Maybe more recent versions of C++ cater for testing the availability of input?

Note: another solution idea I read was to view stdin as a socket, and then to use select and poll to test whether there is pending data. If I recall well, the person suggesting this suggested that it may not work as desired on every platform.

#4 - 29 Aug 2014 12:05 - John Abbott

- Status changed from New to In Progress

- Target version changed from CoCoA-5.1.1 Seoul14 to CoCoA-5.1.2 summer 2015

- % Done changed from 0 to 10

Since C++ is rather less than helpful in this matter, I think a better solution would be to add a command(s?) to the CoCoA interpreter which suppress printing of the prompt, at least for a region of the input.

A simple approach might be an alternative block...endblock command which prints no prompts internally. Or we could just have two commands: NoPrompt and prompt which (de)activate printing of prompts. To avoid risk of accidentally typing in these commands, the command names could contain some "funny characters"...

Note (20140829): other possible names **PromptOn** and **PromptOff**

#5 - 29 Aug 2014 12:30 - John Abbott

I have checked in the code for automatic prompt suppression (in Main.C:153 and LineProviders.C:85; but the code is commented out because I doubt it works portably).

#6 - 11 May 2015 14:37 - John Abbott

- Target version changed from CoCoA-5.1.2 summer 2015 to CoCoA-5.?.?

#7 - 20 Nov 2020 13:35 - John Abbott

- Target version changed from CoCoA-5.?.? to CoCoA-5.4.0

Apparently the ability to suppress prompts could be useful for the ApCoCoA interface.

For that application, it would suffice to have a command line flag which suppresses all prompts.

A potential disadvantage of **PromptOff** would be that a user could "accidentally" send **PromptOn** to CoCoA running inside ApCoCoA, and thereby cause trouble.

Another possible solution could be a command like **SourceString** which opens a string as if it were a file, and then effectively runs the same as source; not altogether sure how error mesgs would work in this case (since they usually refer to filename and line number).

Opinions/Comments?

#8 - 22 Feb 2021 13:31 - John Abbott

- Related to Support #911: CoCoA-5 prompt for incomplete input added

#9 - 22 Feb 2021 17:21 - John Abbott

- Assignee set to John Abbott

- % Done changed from 10 to 50

I have added a command line option **--no-prompt** which forces the function which produces the prompt string to return an empty string (always).

A quick test suggests that it works.
I do a few more tests, and some cleaning. Then check-in.

#10 - 02 Mar 2021 22:54 - John Abbott

If it is possible to call getline with a timeout (*e.g.* 0.1s) then we could do the following:

- do not print any prompt (initially)
- call getline with short timeout (say 0.1s)
- if no input, then print the prompt, and call getline again (but with no timeout)
- otherwise if there is a new input line, do not print the prompt, just process the new line

This will produce a slightly odd effect with normal interactive use: the prompt will appear with a very slight delay (and someone who types very fast may be able to type some chars before the prompt appears -- that's anyway possible if the previous command was slow).

Is it worth trying this? (assuming getline with timeout exists)

#11 - 12 Mar 2021 09:36 - John Abbott

There is a potentially useful code outline on StackOverflow (https://stackoverflow.com/questions/15524122/how-to-implement-timeout-for-getline):
[WARNING: BUGGY CODE -- has dangling ref]

```
#include <thread>
#include <atomic>
#include <iostream>
#include <string>

int main()
{
    std::atomic<bool> flag = false;
    std::thread([&]
    {
        std::this_thread::sleep_for(std::chrono::seconds(5));

        if (!flag)
            std::terminate();
    }).detach();

    std::string s;
    std::getline(std::cin, s);
    flag = true;
    std::cout << s << '\n';
```

#12 - 12 Mar 2021 12:21 - John Abbott

I have just tried a minor variant of the code excerpt given in the previous comment (11).
It did not work (as expected).

The very first prompt did not appear at all.
After sending several lines (via cut-and-paste), a prompt was printed at the end of the (3) lines... good!
BUT after typing in just 1 more line, suddenly 4 = 3+1 prompts were printed out... bad :-(

I wonder if the flag should be global (via a shared_ptr)?

#13 - 12 Mar 2021 17:40 - John Abbott

- Status changed from In Progress to Resolved

- % Done changed from 50 to 70

I now have a prototype which works as expected in various tests :-)

One question: if I start CoCoA inside emacs by sending a line (say), should the very first prompt be printed?
Currently the very first prompt is not printed since the criterion for printing a prompt is that no input line was received within 0.25s (current waiting time).

What do you think?

PS seems OK also when running in terminal with --no-readline flag; if readline is active, it handles things differently (but still acceptable behaviour).

#14 - 15 Mar 2021 10:47 - John Abbott

The current may have a weak spot: if very many simple lines are given as input, presumably the process could create very many concurrent threads since the first thread will vanish only after 0.25s (by which time there may be another 1000 threads... if the input lines are very simple).

I have tried a test with input like the following:

```
t0 := CpuTime();
A := 1;
A := 1;
...
A := 1;
TimeFrom(t0);
```

So far the current impl of my linux computer works well enough with 1000 copies of A := 1;

I see two possible remedies:

1. reduce the wait time from 0.25s (how long a line can be read in 0.25s?)
2. change the sleep call into a 25-iter loop which sleeps for 0.01s and then checks the flag

I suppose approach 1 is more KISS...

UPDATE my computer can read a line about 800k long in 0.25s (the line was just many repeats of A := 1;)
How short a time can a thread "sleep" for?

#15 - 16 Mar 2021 09:44 - John Abbott

I have tried to cause problems by giving inputs with many simple lines, but observed no indication of trouble (surely there must have been several thousand threads...?)

One useful change was to avoid creating a thread if **--no-prompt** was set to true: suddenly the test ran about 10 times faster (wow!)

Mu experience suggests that 0.25s wait time is a bit too long... it feels odd that the prompt "hesitates" before appearing.
I'll try 0.1s.

#16 - 16 Mar 2021 11:52 - John Abbott

I finally managed to make the orig impl fail (with system exception Resource temporarily unavailable).
There were 100000 copies of A := 1; and the time delay was 0.25s. There was no problem with a delay of 0.1s.

Changing the wait into a loop fixed the problem:

```
  for (int slept = 0; slept < SNOOZE_TIME; slept += 10)
  {
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
    if (*GetlineCompleted_ptr) return;
  }
```

Strictly the loop should use a steady clock to check the actual amount of time that has been slept...

#17 - 16 Mar 2021 12:11 - John Abbott

The loop version from the previous comment worked OK, but incurred a noticeable run-time overhead.

For the time being I have gone back to a single sleep_for with a time of 200ms. That worked on my computer (even with 100000 lines of input).
This seems to be a reasonable compromise... but the "resource unavailable" exception could haunt us on a different platform?

#18 - 17 Mar 2021 11:21 - John Abbott

I confirm that disappointingly large overhead mentioned in comment 15 above.

```
./cocoa5 --no-readline < TEST.cocoa5             # takes about 3.6s
./cocoa5 --no-readline --no-prompt < TEST.cocoa5 # takes about 0.2s
```

I am now considering changing the design so that just 2 threads are used: main thread, and a separate one which decides whether to print a prompt.
Communication would have to be via globals... hmmm, that's not a good sign.

#19 - 17 Mar 2021 11:24 - John Abbott

There could be a subtle bug to deal with: what happens if the interpreter exits while the prompt thread is still running? Is the impl safe in this situation?

#20 - 14 May 2021 17:43 - John Abbott

- Status changed from Resolved to Closed

- % Done changed from 70 to 100

- Estimated time set to 7.47 h

The latent problem did appear (see issue #1594; now fixed).
A different design might be better, but this is good enough for now... maybe we will revise it later.
Closing.

NOTE: this works only when readline is DISABLED