plainspooky / inliner

master | Go to file | Add file | Code

plainspooky Added task to auto... ... on 4 Aug 2018 25

contrib	Added task to automatically...	2 years ago
examples	Adding examples's pages	3 years ago
images	tests organized, created a p...	3 years ago
tests	tests organized, created a p...	3 years ago
.gitignore	tests organized, created a p...	3 years ago
Inliner.py	locale changes removed	3 years ago
README.md	tests organized, created a p...	3 years ago
inliner	locale changes removed	3 years ago
requiremen...	initial commit	3 years ago

README.md

Inliner is a tool that converts a BASIC program written in an indented BASIC style into the numbered lines style. It supports labels, definitions and statements in multiple lines and you could to choose the comments that will be exported after conversion.

How it works?

About

A tool written in Python that converts indented BASIC programs into a numbered lines style.

#python
#code-generation
#basic #msx

Readme

Releases

No releases published

Packages

No packages published

Languages

- Python 89.3%
- Visual Basic .NET 10.7%

Put each statement of your BASIC program in a line and let **Inliner** organize them in the same numbered line. Leave an empty line between statements to force a new numbered line creation. You could split BASIC statements in more than one lines and use a backslash -- `\` -- to join them in a single line. For more information and details, check out at "./samples" directory.

How to use

Type your program using your preferred text editor. Like this `example.bas` here:

```
input "What is your name?";k$

print "Hello ";k$;"!"
```

Open a terminal and type `inliner example.bas` and the output will be:

```
10 input "What is your name?";k$
20 print "Hello ";k$;"!"
```

So, transfer this converted code to the target computer/emulator and run it!

Parameters

There are parameters to help to customize the output, one of these is `--start` that defines the number of the first line:

```
$ inliner example.bas --start=100
100 input "What is your name?";k$
110 print "Hello ";k$;"!
```

The other is the `--step` that sets the line's increment:

```
$ inliner example.bas --step=5
10 input "What is your name?";k$
```

```
15 print "Hello ";k$;"!"
```

They could be used together:

```
$ inliner example1.bas --start=2000 --step=5
2000 input "What is your name?";k$
2005 print "Hello ";k$;"!"
```

And the default value of both parameters are 10.

Features

There are some features in **Inliner** that help to improve the BASIC programming.

Comments

There are two kinds of comments here, one is temporary (will be removed during conversion process) and the other is persistent (will be included in the converted program). To create temporary comments use a single apostrophe -- `'` -- and to create a persistent one use two apostrophes -- `''` -- (or the BASIC statement `REM`).

Example:

```
'' EXAMPLE.BAS

' get user's name
input "What is your name?";k$

' print name on screen
print "Hello ";k$;"!"
```

Will result:

```
10 ' EXAMPLE.BAS
20 input "What is your name?";k$
30 print "Hello ";k$;"!"
```

Line splitting

You could split a statement in two or more lines. Just use a backslash -- `\` -- at begining of line to mark them as part of the previous statement.

Example:

```
'' EXAMPLE.BAS

' get user's name
input "What is your name?";k$

' print name on screen
print "Hello ";
    \ k$;",
    \ how do you doing?"
```

Will result:

```
10 ' EXAMPLE.BAS
20 input "What is your name?",k$
30 print "Hello "; k$;", how do you doing?"
```

Labels

Labels are logical marks that points to specific parts of the program. They can contains uppercase and lowercase letters, numbers and the underline -- `_` --. They must be defined in its own line and finished using a comma -- `:` . To use a label as part of a statement put it between `{{ }}` .

```
'' EXAMPLE.BAS

main_loop:
    ' get user's name
    input "What is your name?";k$

    ' print name on screen
    print "Hello ";
        \ k$;",
        \ how do you doing?"
```

```
    ask_loop:
        input "Again (Y/N)?";k$
        if k$="N" then
            \ end

        if k$="Y" then
            \ goto {{main_loop}}

        goto {{ask_loop}}
```

Will result:

```
10 ' EXAMPLE.BAS
20 input "What is your name?";k$
30 print "Hello "; k$;", how do you doing?"
40 input "Again (Y/N)?";k$:if k$="N" then end
50 if k$="Y" then goto 20
60 goto 40
```

And there is an special label, the `@` (self), thats makes reference to the statement's own numbered line.

```
i=1
print "counting... ";

print i;
i=i+1
goto {{@}}
```

Will result:

```
10 i=1:print "counting... ";
20 print i;:i=i+1:goto 20
```

Definitions

And for last, there is the definiion. It is a special kind of label that stores values or even short sequences of statements (like a macros). To put a definition in your program create a temporary label begining with the keyword `define` :

```
' define «name of definition» «value of
```

```
definition»
```

Use a definition as tha same way of a label, puting its name between `{{ }}` .

Example:

```
'' EXAMPLE.BAS
' define YES k$="Y" or k$="y"
' define NO k$="N" or k$="n"

main_loop:
    ' get user's name
    input "What is your name?";k$

    ' print name on screen
    print "Hello ";
          \ k$;",
          \ how do you doing?"

    ask_loop:
        input "Again (Y/N)?";k$
        if {{NO}} then
            \ end

        if {{YES}} then
            \ goto {{main_loop}}

        goto {{ask_loop}}
```

Wil result:

```
10 ' EXAMMPLE.BAS
20 input "What is your name?";k$
30 print "Hello "; k$;", how do you doing?"
40 input "Again (Y/N)?";k$:if k$="N" or k$="n"
then end
50 if k$="Y" or k$="y" then goto 20
60 goto 40
```