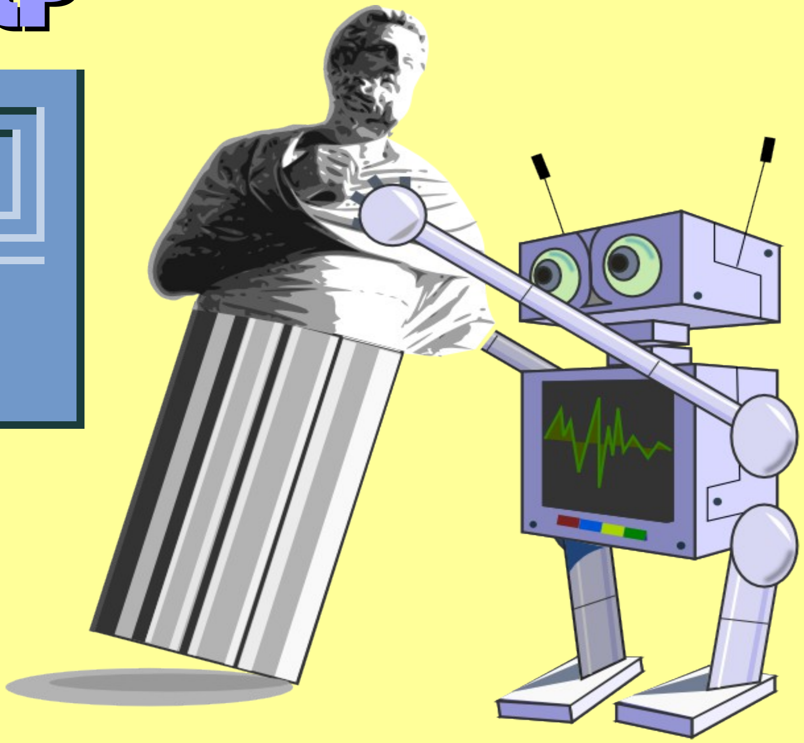
NANOLP



Introduction to LP

LP : **Literate Programming** -
more reliable, readable, supportable

Benefits of LP

- Makes programming fine!
- Is good for children teaching
- Is good for books, articles...
- Is good for collaborative working
- Makes software supportable
- Makes code readable (and printable)
- Reduces bugs!

How to achieve this?

try new free tool

NanoLP

History. New terminology

- introduced by Donald Knuth near 1970s
- alternative to the structured programming paradigm
- tools: WEB, CWEB, noweb...

- **chunk** — piece of code
- **command** — macros
- **surrounder** — strings used as left, right surround command symbols
- **fetcher** — method of source retrieving

Problems with old tools

- weird syntax
- only one input files format (based on TeX, LaTeX, XML...)
- presentation facilities are limited by input format
- no publishing facility
- weak code transformation
- not extendible, not scriptable

How it looks

```
To test if number is positive
(pos), use: `n >= 0`. Now
absolute function will be:


    int abs(int n) {
      if ( [[=pos]] )
        return n;
      else return -n;
    }

                         Markdown
```

Simple example

- easy to read
- flow of thoughts
- free structure
- doc formatting
- images, tables, diagrams, links, etc...

How it looks

```
Standard C headers guard
(hguard) is:

  #ifndef ${file}_H

  #define ${file}_H


and `#endif` at the end. So
(file.x, prn.h):

  [[=hguard.0, file:PRN]]

  #define PRN(x) printf(x)

  [[=hguard.1]]

                        Markdown
```

Flex macros:

- placeholders
- multipart
- saving to file
- multiple output files
- and more options...

see result ►

How it looks

```
#ifndef PRN_H
#define PRN_H
#define PRN(x) printf(x)
#endif
```

and result will be saved in „prn.h“

How it looks

Standard C headers guard (hguard) is:

```
   #ifndef ${file}_H

   #define ${file}_H
```

and #endif at the end. So (file.x, prn.h):

```
   [[=hguard.0, file:PRN]]

   #define PRN(x) printf(x)

   [[=hguard.1]
```

OpenOffice

WYSIWYG way:

- edit result doc
- no special syntax
- rich styling
- easy for diagrams, tables, etc...

More examples

See **test/tests** folder in source archive...

NanoLP advantages

- Clean, human (book) oriented syntax
- Standard documents formats
- Flex macros system
- Fetching sources via different protocols
- Highly customizable, extendible
- Publishing code on the Web
- Imports

and many more...!

Documents formats

- OpenOffice, LibreOffice
- Markdown
- MultiMarkdown
- ReStructuredText
- Creole
- TeX/LaTeX
- HTML/XML
- AsciiDoc
- Txt2Tags

Work in WYSIWYG office suite or with Wiki tools, or even with publishing systems!

Code sources fetching

- via FTP (authorized too)
- via HTTP (authorized too)
- from ZIP archive (crypted too)
- from shell command output
- ...and sure from local FS

User can add custom fetchers!

Publishing on the Web

1. Generated cross-references file
2. Raw documents (Creole, Markdown, etc)
3. Converted to HTML documents
4. Special published HTML documents

1: Total info, list of chunks, interactive links
4: Converted LP macroses to interactive links

Custom styling is supported

Publishing cons

instead of corporative Wiki with out-of-dated code documentation...

use forever actual documentation with:

- navigation
- source extracting
- source importing
- good for reading, printing, teaching

Syntax concept

- ...command...chunk...
  - define named chunk
- ...sys command...command...chunk...
  - exec sys command, followed by case 1)
- ...command...chunk...chunk...
  - define multipart named chunk

- sys commands are executed

- named chunks are pasted

How it is parsing

extracted:

- commands
- inline chunks
- block chunks

▫ commands in **doc** fragment are surrounded by 2 strings: [[,]] or <<,>> or any user defined

▫ commands in **code** fragment are surrounded by 2 strings: [[,]] or <<,>> or any user defined

▫ pasted commands begins with = symbol

System commands

- import another code sources
- flush result to files
- create variables or maps of them
- catch events...

Code transformation

- paste
- substitute vars
- transform of vars
- transform of chunks
- custom join of chunks
- custom join of vars

Customize via:

- events
- parameters
- positional args
- keyword args

Code transformation

It is possible to paste «mystructs»:

```
[[=mystructs, one, two, three]]
```

and to get:

```
struct mystruct m[] = {
    { ONE, ˝one˝},
    { TWO, ˝two˝},
    { THREE, ˝three˝} };
```

See events catching about it

Other customization

- Surround symbols
- Input format detection
- Parsing options
- Custom styles (via CSS)
- Custom sys commands
- Custom fetchers
- Custom parsers
- Custom event handlers

Details

- Supported UTF-8
- Python 2.7 - 3+ compatible
- Works on Linux, Windows
- Many tests are provided
- Free and open source
- GNU licensed

More info

Project home: http://code.google.com/p/nano-lp/

Blog: http://balkansoft.blogspot.com/